ENCYCLOPEDIA FOR THE TRS-80*

A library of useful information for your TRS-80

Business
Education
Games
Graphics
Hardware
Home Applications
Interface
Tutorial
Utility

VOLUME 10

*Trademark of Tandy Corp.

ENCYCLOPEDIA for the TRS-80*

ENCYCLOPEDIA for the TRS-80*

VOLUME 10

wayne GREEN inc.
PETERBOROUGH NH 03458

*TRS-80 is a trademark of Radio Shack division of Tandy Corp.

The LDOS disk operating system, a product of Logical Systems, was used in the technical production of this book.

FIRST EDITION
FIRST PRINTING SEPTEMBER 1982

Edited by Kate Comiskey and Katherine Lindquist
Proofread by Ann Winsor
Production: Margaret Baker, Gary Ciocci,
Linda Drew, Thomas Villeneuve, Robert M. Villeneuve,
Sandra Dukette, Elizabeth Libby, Karen Stewart
Technical Editor: Jim Heid
Illustrations by Howard Happ

FOREWORD

The Biggest Difference

There are lots of arguments about which computer is the best. The answer to this question lies not in which hardware is best. That is really irrelevant, when you understand the field. The major value of any computer lies in the software and the information available for it. Hence this encyclopedia.

The TRS-80 is by no means the best computer on the market as far as its hardware is concerned, but with the support of *80 Microcomputing* magazine and this encyclopedia series, you have an almost unlimited source of information on how to use your computer—and of programs. With this information source the TRS-80 is by far the most valuable computer system ever built. No other computer, at any price, has anything approaching this amount of user information and programs available.

Most encyclopedias try to freeze everything at one time and are thus able to divide the material up alphabetically. This is a new kind of encyclopedia —a living one—with each new volume keeping you up to date on the very latest information on using your computer and the newest of programs.

Your computer can be a fantastic teaching device, a simulator, a way to play all sorts of fascinating games, a business aid, a scientific instrument, a control unit for machinery. . . . It is one of the most flexible gadgets ever invented. All of these applications are possible *if* you have the information and the programs. This encyclopedia will give you these.

To get the best use of your TRS-80, don't miss a single volume of the *Encyclopedia for the TRS-80.*

WAYNE GREEN
Publisher

CONTENTS

Please note: Before typing in any listing in this book, see Appendix A.

Encyclopedia Loader™

The editors of Wayne Green Books want to help you maximize your microcomputing time, so they created the **Encyclopedia Loader™**.

The **Encyclopedia Loader** is a special series of cassettes that offer the longer programs in the **Encyclopedia for the TRS-80*** in ready-to-load form. Each of the ten volumes of the Encyclopedia provides the essential documentation for the programs on the Loader.

With the **Encyclopedia Loader**, you'll save hours of keyboard time and eliminate the aggravating search for typos. The **Encyclopedia Loader** for Volume 10 will contain the programs for the following articles:

Check Storage
Loan Amortization
Plan Ahead
Physics in Motion
SBLOCK
Planning Your Retirement
Lazy Logic Trainer
Modifying Scripsit to Send Control Characters to Printers
Shortstuff

Encyclopedia Loader™ for Volume 1	**EL8001**	**$14.95**
Encyclopedia Loader™ for Volume 2	**EL8002**	**$14.95**
Encyclopedia Loader™ for Volume 3	**EL8003**	**$14.95**
Encyclopedia Loader™ for Volume 4	**EL8004**	**$14.95**
Encyclopedia Loader™ for Volume 5	**EL8005**	**$14.95**
Encyclopedia Loader™ for Volume 6	**EL8006**	**$14.95**
Encyclopedia Loader™ for Volume 7	**EL8007**	**$14.95**
Encyclopedia Loader™ for Volume 8	**EL8008**	**$14.95**
Encyclopedia Loader™ for Volume 9	**EL8009**	**$14.95**
Encyclopedia Loader™ for Volume 10	**EL8010**	**$14.95**

(Please add $1 50 per package for postage & handling)

Mail your order to "Encyclopedia Loader Sales," Wayne Green Books, Pine Street, Peterborough, NH 03458 or call **(1-800-258-5473)**.

*TRS-80 is a trademark of Radio Shack Division of Tandy Corp

BUSINESS

Check Storage Program
Loan Amortization
Plan Ahead—A Program for Project Planning

Check Storage Program

by Dan Keen and Dave Dischert

This program will create and maintain a file of checks. The information stored on each check is:

1) Check number (up to five digits are allowed).
2) Date (up to eight digits. Use the format MM/DD/YY).
3) Payee, that is, pay to the order of. . . (up to 30 characters can be entered).
4) Amount of check (the limit is $99,999.99).
5) Remarks, reminders, or other notes. 30 characters can be stored.

This program will run on the TRS-80 Model I or III with no changes necessary. It will store about 1000 checks on the Model III computer. Only one disk drive and a minimum of 32K of RAM are required.

Menu Options

The main menu displays the options as shown in Figure 1.

```
CHECK STORAGE PROGRAM

<A>DD A CHECK
<S>EARCH / DELETE / MODIFY
<T>OTALS FOR THE MONTH
<P>RINT LIST OF ALL CHECKS IN STORAGE
<Q>UIT
```

Figure 1. *Menu options*

Adding a Check to the File

Select the <A>DD A CHECK option by pressing the A key. Next, you will be asked to enter the check number, the date, whom the check was made out to, the amount, and any remarks or comments you may wish to include. The date must be in the form of two digits for the month, two for the day, and

Printing out all of the Checks on File

The <P>RINT LIST OF ALL CHECKS IN STORAGE option will show you, either on the video display or on a printer, every check currently on file. You must answer Y or N to the question:

DO YOU WANT A PAPER PRINTOUT? (Y/N).

This gives you a chance to type the entry in again if you made a mistake. Simply press the Y or N keys. It isn't necessary to press the ENTER key following your selection. Even if you were to answer <Y>ES and the check was stored on the diskette, you could still change any parameter later by using the <S>EARCH / <D>ELETE / <M>ODIFY option from the main menu.

After you have typed in the check and indicated that the information is correct, you will again be asked to enter the number of the next check. In this way you can continue adding checks until you have entered all that you care to store. To return to the main menu, simply respond to the ENTER CHECK NUMBER question by pressing the ENTER key. This option is displayed on top of the screen to remind you.

Search, Delete, or Modify

By choosing the <S>EARCH function, you can look up a check by identifying either the check number or the name of the payee. Once you have found the check you are looking for, the computer will display all of the parameters associated with that check and ask what you want to do next:

<D>ELETE
<M>ODIFY
<R>ETURN TO MAIN MENU

At this point you could delete that check from the list by pressing the D key. To change either the check number, date, payee, amount, or remarks, select M.

You will be asked to enter the correct check number, date, payee, amount, and remarks. If there is no change on a particular item, simply respond by pressing the ENTER key and the old information for that category will be retained. This procedure is explained on the screen as you go through it.

Getting Totals for a Particular Month

You can find out the total of all checks written for any particular month instantly, simply by selecting the <T>OTALS FOR THE MONTH option. You will be asked:

WHAT MONTH DO YOU WANT THE TOTAL OF (ENTER 2 DIGITS — MM)?

January would be 01, February would be 02, December would be 12, and so on. Then you are asked whether or not you would like a paper printout of the checks and the total. Select Y for yes, N for no.

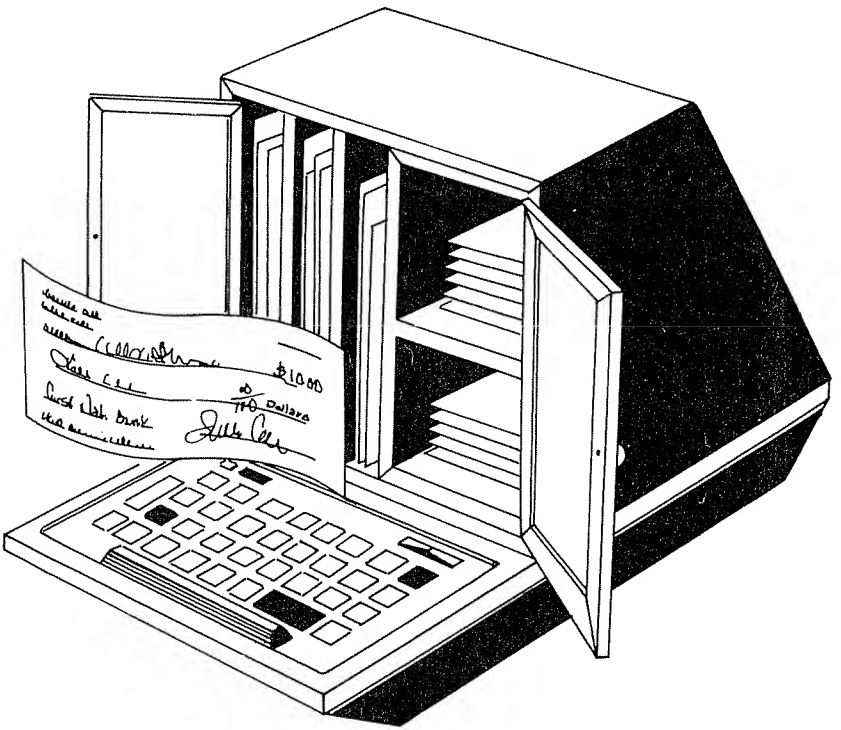
two for the year. For example, January 18, 1982 would be entered as 01/18/82.

When you have entered this information, the bottom of the screen will prompt you with:

<Y>ES THIS INFORMATION IS CORRECT
<N>O—I WANT TO RETYPE THIS ENTRY

Ending the Program

When you are done, press Q to quit. At that point all files are closed, and it is safe to remove the diskette.



Program Listing. *Check storage*

Encyclopedia Loader

```
0 ' CHECK STORAGE PROGRAM

2 CLS:CLEAR1000:M$="$$##,###.##":L=15617:PRINT@21,"CHECK STORAGE PROGR
    AM":PRINTSTRING$(64,"=")
4 PRINT@384,"<A>DD A CHECK":PRINT"<S>EARCH / DELETE / MODIFY":PRINT"<T
    >OTALS FOR THE MONTH":PRINT"<P>RINT LIST OF ALL CHECKS IN STORAGE
    ":PRINT"<Q>UIT"
5 IK$=INKEY$:IFIK$="A"THEN10ELSEIFIK$="C"THEN2000ELSEIFIK$="S"THEN3000
    ELSEIFIK$="T"THEN8000ELSEIFIK$="P"THEN9000ELSEIFIK$="Q"THENEND
6 A=PEEK(L):POKEL,32:FORX=1TO40:NEXT:POKEL,A:L=L+64:IFL>16128THENL=156
    17
7 GOTO5
10 GOSUB10000
34 GET1,1
36 R%=VAL(RE$):SR%=VAL(SU$):IFR%=0THENR%=1
40 GET1,R%
80 CLS:PRINT@8,"JUST HIT <ENTER> IN RESPONSE TO 'CHECK NUMBER' TO QUIT
    "
85 PRINT@192,"ENTER";:PRINT@198,"> CHECK NUMBER          ";:LINEINPUTA$
    :PRINT@192,"     ";
90 IFA$=""ANDSR%>0THENFORDE=0TO2:LSETCH$(SR%+DE)=STRING$(5,32):LSETDA$
    (SR%+DE)=STRING$(8,32):LSETPA$(SR%+DE)=STRING$(30,32):LSETAM$(SR%
    +DE)=STRING$(7,32):LSETRE$(SR%+DE)=STRING$(30,23):NEXTDE:PUT1,R%
95 IFA$=""THENPUT1,R%:GET1,1:LSETRE$=STR$(R%):LSETSU$=STR$(SR%):PUT1,1
    :A$="":B$="":C$="":D$="":E$="":CLOSE:RUN
100 PRINT@256,"ENTER";:PRINT@262,"> DATE (MM/DD/YY)       ";:LINEINPUTB
    $:PRINT@256,"     ";
120 PRINT@320,"ENTER";:PRINT@326,"> PAY TO THE ORDER OF  ";:LINEINPUTC
    $:PRINT@320,"     ";
140 PRINT@384,"ENTER";:PRINT@390,"> AMOUNT  (NO COMMAS) $";:LINEINPUTD
    $:PRINT@384,"     ";
160 PRINT@448,"ENTER";:PRINT@454,"> REMARKS               ";:LINEINPUTE
    $:PRINT@448,"     ";
190 PRINT@840,"<Y>ES THIS INFORMATION IS CORRECT
       <N>O - I WANT TO RETYPE THIS ENTRY"
195 IK$=INKEY$:IFIK$="Y"THEN200ELSEIFIK$="N"THENA$="":B$="":C$="":D$="
    ":E$="":GOTO80ELSE195
200 LSETCH$(SR%)=A$:LSETDA$(SR%)=B$:LSETPA$(SR%)=C$:LSETAM$(SR%)=D$:LS
    ETRE$(SR%)=E$
300 SR%=SR%+1:IFSR%>2THENSR%=0:GOTO350ELSE360
350 PUT1,R%:R%=R%+1:FORXX=0TO2:LSETCH$(XX)=STRING$(5,32):LSETDA$(XX)="
    ":LSETPA$(XX)="":LSETAM$(XX)="":LSETRE$(XX)="":NEXT
360 A$="":B$="":C$="":E$="":D$="":I=0:GOTO80
3000 '
                *** SEARCH / DELETE / MODIFY ***

3030 GOSUB10000:CLS:PRINT@20,"SEARCH / DELETE / CHANGE":PRINT:PRINT:PR
    INT"WHAT DO YOU WANT TO SEARCH BY?
<C>HECK NUMBER
<P>AYEE"
3032 IK$=INKEY$:IFIK$="C"THEN3038ELSEIFIK$="P"THEN3034ELSE3032
3034 GET1,1:RK%=VAL(RE$):INPUT"WHAT IS THE PAYEE NAME ";Z$
3035 FORR%=1TORK%:GET1,R%:FORSR%=0TO2:IFZ$=LEFT$(PA$(SR%),LEN(Z$))THEN
    3100
3036 NEXTSR%,R%:PRINT"** NAME NOT FOUND ***":FORDE=1TO1000:NEXT:CLOSE:
    RUN
3038 GET1,1:RK%=VAL(RE$):INPUT"WHAT IS THE NUMBER OF THE CHECK YOU WAN
    T TO SEARCH FOR ";Z$
3040 FORR%=1TORK%:GET1,R%:FORSR%=0TO2:IFZ$=LEFT$(CH$(SR%),LEN(Z$))THEN
    3100
3050 NEXTSR%,R%:PRINT"** CHECK NUMBER NOT FOUND **":FORDE=1TO1000:NEXT
    :CLOSE:RUN
3100 CLS:PRINT@26,"LOCATED CHECK":PRINT@128,"CHECK #     ";CH$(SR%)
3110 PRINT@192,"DATE        ";DA$(SR%):PRINT@256,"PAYEE       ";PA$(SR
    %)
3120 PRINT@320,"AMOUNT      ";AM$(SR%):PRINT@384,"REMARKS     ";RE$(SR
    %)
```

```
3130 PRINT@832,"<D>ELETE":PRINT"<M>ODIFY":PRINT"<R>ETURN TO MENU";
3140 A$=INKEY$:IFA$="D"THEN3150ELSEIFA$="M"THEN4000ELSEIFA$="R"THENCLO
     SE:RUNELSE3140
3150 '                DELETE  SUBROUTINE
3170 LSETCH$(SR%)="*****":LSETDA$(SR%)=STRING$(8,32):LSETPA$(SR%)=STRI
     NG$(30,32):LSETAM$(SR%)=STRING$(7,32):LSETRE$(SR%)=STRING$(30,128
     )
3180 PUT1,R%:CLOSE:RUN
4000 '                MODIFY  SUBROUTINE
4010 PRINT@512,CHR$(31);:PRINT@512,"(JUST HIT <ENTER> IF THERE IS NO C
     HANGE ON ANY ITEM)"
4011 LINEINPUT"ENTER CORRECT CHECK # ";A$:IFA$=""THENA$=CH$(SR%)
4012 LINEINPUT"ENTER CORRECT DATE    ";B$:IFB$=""THENB$=DA$(SR%)
4013 LINEINPUT"ENTER CORRECT PAYEE   ";C$:IFC$=""THENC$=PA$(SR%)
4014 LINEINPUT"ENTER CORRECT AMOUNT  ";D$:IFD$=""THEND$=AM$(SR%)
4017 LINEINPUT"ENTER CORRECT REMARKS ";E$:IFE$=""THENE$=RE$(SR%)
4020 LSETCH$(SR%)=A$:LSETDA$(SR%)=B$:LSETPA$(SR%)=C$:LSETAM$(SR%)=D$:L
     SETRE$(SR%)=E$
4030 PUT1,R%:CLOSE:RUN
6000 REM
        INKEY SUBROUTINE

6010 IK$=INKEY$:IFIK$=""THEN6010ELSERETURN
8000 REM
           TOTALS FOR THE MONTH

8010 CLS:TL=0
8020 INPUT"WHAT MONTH DO YOU WANT THE TOTAL OF (ENTER 2 DIGITS - MM)";
     MO$
8030 PRINT"DO YOU WANT PRINTER OUTPUT? <Y>ES  OR  <N>O"
8040 GOSUB6000:IFIK$="Y"THENSW=1ELSEIFIK$="N"THENSW=0ELSE8040
8041 IFSW=1THENLPRINT"CHECK #  DATE         TO WHOM
               AMOUNT     REMARKS":LPRINTSTRING$(77,"-")
8050 CLS:GOSUB10000:GET1,1:R%=VAL(RE$):IFR%=0THENPRINT"NO CHECKS HAVE
     BEEN STORED":FORDE=1TO1000:NEXT:CLOSE:RUN
8060 FORX%=1TOR%:GET1,X%:FORSR%=0TO2
8070 IFMO$=LEFT$(DA$(SR%),2)THENGOSUB8500
8080 NEXTSR%,X%:PRINT"TOTAL FOR THIS MONTH =";USINGM$;TL
8090 IFSW=1THENLPRINT" ":LPRINT"TOTAL FOR THIS MONTH =";USINGM$;TL:FOR
     DE=1TO10:LPRINT" ":NEXT
8100 PRINT:LINEINPUT"HIT THE <ENTER> KEY TO RETURN TO MENU";IK$:CLOSE:
     RUN
8500 IFLEFT$(CH$(SR%),5)="*****"THENRETURN
8510 TL=TL+VAL(AM$(SR%)):PU=VAL(AM$(SR%)):PRINTCH$(SR%);"    ";DA$(SR%)
     ;"    ";PA$(SR%);"    ";USINGM$;PU
8520 IFSW=1THENLPRINTCH$(SR%);"    ";DA$(SR%);"    ";PA$(SR%);"    ";U
     SINGM$;PU;:LPRINT"    ";RE$(SR%)
8530 RETURN
9000 REM
           PRINT LIST OF ALL CHECKS ON DISKETTE

9010 GOSUB10000:CLS:C=0
9020 PRINT"DO YOU WANT PRINTER OUTPUT?  <Y>ES  OR  <N>O"
9030 GOSUB6000:IFIK$="Y"THENGOTO9100ELSEIFIK$="N"THEN9200ELSE9030
9100 GET1,1:R%=VAL(RE$)
9105 LPRINT"CHECK #  DATE         TO WHOM
     AMOUNT      REMARKS":LPRINTSTRING$(74,"-")
9110 FORX%=1TOR%:GET1,X%:FORSR%=0TO2
9120 IFLEFT$(CH$(SR%),5)="*****"THENGOTO9140
9130 PU=VAL(AM$(SR%)):LPRINTCH$(SR%);"    ";DA$(SR%);"    ";PA$(SR%);"
          ";USINGM$;PU;:LPRINT"    ";RE$(SR%)
9140 NEXTSR%,X%:FORDE=1TO10:LPRINT" ":NEXT:CLOSE:RUN
9200 GET1,1:R%=VAL(RE$):CLS:COUNTER=0
9210 FORX%=1TOR%:GET1,X%:FORSR%=0TO2
9220 IFLEFT$(CH$(SR%),5)="*****"THENGOTO9240
9230 PU=VAL(AM$(SR%)):PRINTCH$(SR%);"  ";DA$(SR%);"  ";PA$(SR%);"    "
     ;USINGM$;PU:COUNTER=COUNTER+1
9235 IFCOUNTER>10THENPRINT@900,"HIT THE <ENTER> KEY TO SEE MORE":GOSUB
     6000:CLS:COUNTER=0
9240 NEXTSR%,X%:PRINT"HIT THE <ENTER> KEY TO RETURN TO MENU":CLOSE:GOS
     UB6000:RUN
```

Program continued

```
10000 '
            OPEN/FIELD SUBROUTINE

10010 CLOSE:OPEN"R",1,"CHECK/LST":FORSR%=0TO2
10020 FIELD1,SR%*80AS PH$(SR%),5AS CH$(SR%),8AS DA$(SR%),30AS PA$(SR%)
     ,7AS AM$(SR%),30AS RE$(SR%):NEXT
10030 FIELD1,250ASDUMMY$,2AS SU$,3AS RE$:RETURN
```

Loan Amortization

by Dan Keen and Dave Dischert

Loan amortization is needed by many types of businesses. Until the advent of microcomputers, many businesses had to send away to big firms with large computers in order to get a printout of a loan amortization schedule.

This short program will allow you to rapidly create a schedule in your own office. It differs from other loan amortization programs in that it is extremely user oriented. If you can read the screen, you can run the program. The program will work on a Model I or a Model III TRS-80 with any amount of memory. A printer is optional.

Each payment shows the amount that is being paid toward the principal and the amount which is paid as interest. The new outstanding balance is shown also. At the end of each year an itemized summary is given. The computer will show the amount of interest paid during that year, the amount of interest paid since the loan began, the amount of principal paid during the year, and the amount of principal paid since the loan began.

How to Use the Program

When you run the program, you will first be asked to enter the amount of the loan. Do not use commas or dollar signs in your response. A loan for $30,000 would be entered as 30000. Next, you are prompted to enter the number of years you will take to pay off the loan. Finally, you must enter the annual interest rate. If the interest rate is 15 percent, enter the number 15, and if the rate is nine and a half percent enter 9.5.

The computer will calculate the total number of payments, assuming one payment each month, and the amount of each monthly payment.

You are given the option of hard copy, that is, a paper printout, with the question DO YOU WANT A PAPER PRINTOUT? <Y>ES OR <N>O. Simply press the Y or N key on the computer. It isn't necessary to press the ENTER key after this entry is made.

If the print routine is selected, the printer will immediately create the entire amortization schedule. If that option is not selected, the video display will show only one year at a time. This gives you time to study the results before moving on to the next year. Simply press the ENTER key to see the next year. The yearly totals are given on both types of printouts. Figure 1 shows a sample run.

```
ENTER AMOUNT OF LOAN $? 25000
ENTER NUMBER OF YEARS LOAN IS FOR? 20
ENTER ANNUAL INTEREST RATE (IN DECIMAL FORM) ? 9.75
DO YOU WANT PRINTER OUTPUT? <Y>ES OR <N>O  N
```

PAYMENT NUMBER	INTEREST PAID	PRINCIPAL REPAID	OUTSTANDING PRINCIPAL
1	203.13	34.01	24966.00
2	202.85	34.28	24931.70
3	202.57	34.56	24897.20
4	202.29	34.84	24862.30
5	202.01	35.12	24827.20
6	201.72	35.41	24791.80
7	201.43	35.70	24756.10
8	201.14	35.99	24720.10
9	200.85	36.28	24683.80
10	200.56	36.57	24647.20
11	200.26	36.87	24610.40
12	199.96	37.17	24573.20

```
TOTAL INTEREST PAID SINCE LOAN BEGAN $ 2,418.76
TOTAL INTEREST PAID THIS YEAR $ 2,418.76
TOTAL PRINCIPAL PAID SINCE LOAN BEGAN $ 426.80
TOTAL PRINCIPAL PAID THIS YEAR $ 426.80
```

Figure 1. *Sample run*

Program Listing. *Loan Amortization*

Encyclopedia Loader™

```
0 REM MODEL I/III VERSION  UPDATED 05/10/82
10 CLEAR100:CLS:PRINTTAB(20)"HOME LOAN AMORTIZATION":PRINTTAB(26)"VERS
    ION 3.0":PRINTSTRING$(64,"=")
20 INPUT"ENTER AMOUNT OF LOAN (NO COMMAS, PLEASE) $";P
30 INPUT"ENTER NUMBER OF YEARS LOAN IS FOR";Y
40 INPUT"ENTER ANNUAL INTEREST RATE  ( 15% WOULD BE ENTERED  15 )";I1:
    I1=I1/100
60 N=12*Y:I=I1/12:N1=N/2:Q=(1-(1+I)[-N1)/I:V=(1+I)[-N1
70 S=Q*V:W=Q+S:M=P/W
80 PRINT:PRINT"DO YOU WANT PRINTER OUTPUT?  <Y>ES  OR  <N>O"
90 IK$=INKEY$:IFIK$="Y"THENSW=1ELSEIFIK$="N"THENSW=0ELSE90
100 PRINT:PRINT"TOTAL NUMBER OF PAYMENTS=";N
110 A$="$$#,###.##":M$="$$###,###.##":PRINT"MONTHLY PAYMENT=";USINGA$;
    M
120 IFSW=1THENLPRINT"LOAN AMORTIZATION FOR";USINGM$;P
125 IFSW=1THENLPRINT"TOTAL NUMBER OF PAYMENTS=";N
126 IFSW=1THENLPRINT"MONTHLY PAYMENT=";USINGA$;M
130 IFSW=1THENLPRINTY;" YEARS AT ANNUAL INTEREST RATE OF";I1*100;"%":L
    PRINT" "
140 PRINT:PRINT"HIT ANY KEY TO SEE AMORTIZATION SCHEDULE":GOSUB430
150 CLS
160 F$="###             #,###.##          #,###.##          ###,###.##"
170 GOSUB380
180 FP=0:S=0
190 FORZ=1TON
200 T=P*I:U=M-T:P=P-U
210 S=S+T:TI=TI+T:PY=PY+U:PB=PB+U
220 PRINTUSINGF$;Z,T,U,P:IFSW=1THENLPRINTUSINGF$;Z,T,U,P
230 FP=FP+1
240 IF FP=12THEN GOSUB270
250 NEXTZ
260 END
270 PRINT"INTEREST TO DATE ";USINGM$;S;:PRINT"   INTEREST THIS YEAR ";
    USINGM$;TI
280 PRINT"PRINCIPAL TO DATE";USINGM$;PB;:PRINT"   PRINCIPAL THIS YEAR"
    ;USINGM$;PY
290 IFSW=1THENLPRINT"TOTAL INTEREST PAID SINCE LOAN BEGAN ";USINGM$;S
300 IFSW=1THENLPRINT"TOTAL INTEREST PAID THIS YEAR ";USINGM$;TI
310 IFSW=1THENLPRINT"TOTAL PRINCIPAL PAID SINCE LOAN BEGAN ";USINGM$;P
    B
320 IFSW=1THENLPRINT"TOTAL PRINCIPAL PAID THIS YEAR ";USINGM$;PY
325 PY=0:TI=0
330 IFSW=1THENLPRINT" ":LPRINT" ":GOTO360
340 PY=0:TI=0
350 PRINTTAB(16)"HIT ANY KEY TO SEE THE NEXT YEAR";:GOSUB430
360 FP=0:CLS:GOSUB380
370 RETURN
380 PRINT"PAYMENT #       INTEREST          PRINCIPAL        BALANCE DUE
    "
400 IFSW=1THENLPRINT"PAYMENT         INTEREST        PRINCIPAL        OUT
    STANDING"
410 IFSW=1THENLPRINT"NUMBER            PAID           REPAID           PR
    INCIPAL"
420 RETURN
430 IK$=INKEY$:IFIK$=""THEN430ELSERETURN
```

Plan Ahead—A Program for Project Planning

by James N. Devlin

One of the tools used in large industries to aid in the administration of complex projects that involve considerable resources in terms of personnel, machines, and time is a method of planning called PERT. The letters stand for project evaluation and review technique.

This technique was developed for the Navy by the Western Electric Corporation on the Polaris program nearly two decades ago. The original project involved 57 branches with approximately 28,000 individual activities. Even with only a few dozen activities, the use of PERT methods can be of great aid in day-to-day work.

Two major advantages make the use of PERT on a small scale a useful addition to management techniques. The first is improved control over a development or production program. The second is the capacity to distill a large amount of data in a brief, orderly fashion.

PERT gives you, the manager, a useful tool to plan the best possible strategy for applying limited resources to achieve your goal within the available time and cost commitments. Before you can apply PERT, you must have an overall idea of the total project. You must visualize all of the individual tasks necessary to complete the project clearly enough to put them into structured form.

The basic structure of PERT is the *network*. A network is a flow diagram of a plan of action, which displays all the significant events and activities required to complete the given task. The graphic nature of PERT makes it so useful. A brief discussion of the PERT technique itself will enable you to put this program to use immediately to solve your problems of planning. I will then use a real problem to demonstrate its use.

A network is composed of blocks connected by arrows. The blocks are *events*, and the arrows are *activities*. The event is a point in time where a new activity begins and the previous activity ends. The activity is the work done in moving from one event to another. An event number is assigned to each event.

A network is the diagram that links all the parts of a job. Each line in the network represents work (an activity) that must be accomplished. A sketch of the network should be generated as an aid to define all of the work segments. Such a preliminary drawing is shown in Figure 1.

	Activity	Start	Finish	Duration
1	Pre. des. Rel.	1	2	0 days
2	Des. test set	2	3	4
3	Build test set	3	32	15
4	Pre. des. Rel.	1	4	0
5	Build mem proto	4	5	10
6	Des. rel. mem	5	6	0
7	Layout mem bd.	6	7	5
8	Build mem bd.	7	10	8
9	Des. rel. mem	5	8	0
10	Order mem pts	8	9	1
11	Rec. mem pts	9	10	14
12	Assem. mem bds	10	25	5
13	Pre. des. rel.	1	11	0
14	Build cpu proto	11	12	15
15	Des. rel. cpu	12	13	0
16	Layout cpu bd.	13	14	5
17	Build cpu bd.	14	17	8
18	Des.rel. cpu	12	15	0
19	Order cpu pts	15	16	1
20	Rec. cpu pts	16	17	14
21	Assem. cpu bds	17	25	5
22	Pre. des. rel.	1	18	0
23	Build I/O proto	18	19	8
24	Des. rel. I/O	19	20	0
25	Layout I/O bd.	20	21	5
26	Build I/O bd.	21	24	8
27	Des. rel I/O	19	22	0
28	Order I/O pts	22	23	1
29	Rec I/O pts	23	24	14
30	Assem. I/O bds	24	25	5
31	Pre. software spc	1	26	0
32	Des. op sys.	26	27	12
33	Debug op sys	27	30	10
34	I/O software rel	1	28	0
35	I/O & calc	28	29	10
36	Debug I/O & calc	29	30	8
37	Merge software	30	31	10
38	Burn proms	31	32	2
39	Assem. processor	25	32	5
40	Test processor	32	33	10

Table 1. *Activity list*

Using the sketch as a guide, prepare a list of all the tasks that make up the job. They should be arranged in the approximate order or time sequence in which they will occur. This usually takes the form of a laundry list, as shown in Table 1. Naturally, the more detail, the more complex the network. Even

with only a dozen or so tasks, the *critical* path (the one that holds you up the most) may be hidden.

Each bubble in the network is an event. The event is the completion of each activity arrow. Events and activities must be sequenced on a network. No event can be considered complete until all preceding events have been completed.

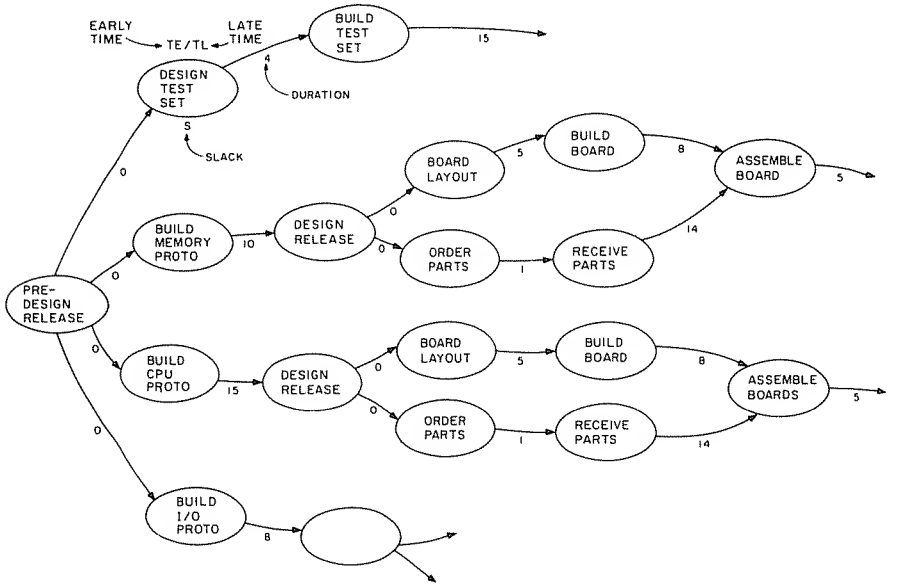


Figure 1. *Preliminary drawing*

Activities (arrows) that have the same predecessor branch out, and activities that have more than one predecessor branch in. The predecessors are those activities which must be finished before a new activity can begin.

After you have listed the activities in some sort of sequential order in the table, you must assign your best estimates of the time it takes to complete each task. This time is called the *duration*. These estimates range from a very careful consideration of previous experience to a carefully thrown dart on a wall calendar. Nevertheless, since the computer has no concept of time, you must assign them the best you can.

Large computers in industry use three estimates of time, usually called optimistic, pessimistic, and normal times. The statistical mean is then calculated. I decided to bypass this complication in favor of being able to change any duration as better estimates become available. It is then a simple matter to rerun the program to obtain an updated output.

The final network is prepared from the information in the table. Begin by drawing the blocks representing the completion of each activity with a line which represents the task between them.

All the events can now be numbered. You should do this in ascending order, from the starting block to the finish block, moving from the top path to the bottom path. Make sure that all branches which terminate in a given bubble have been numbered before you proceed to the next event. This ensures that later events do not occur before the inside branches are completed. The activity time (duration) can be written beneath each arrow. The completed network should resemble the one in Figure 2.

A Sample Application

A typical development project in the computer industry might be the development of a microprocessor board to be used as a controller in some marketable device. Although boards can be purchased, management may decide that for proprietary, cost, or end use reasons to make their own. They ask you to prepare a schedule and estimate the time to complete the project. They may even give you the time to complete and tell you to fit the development into that time frame. Could you do it? Is the time reasonable? Do you have the resources?

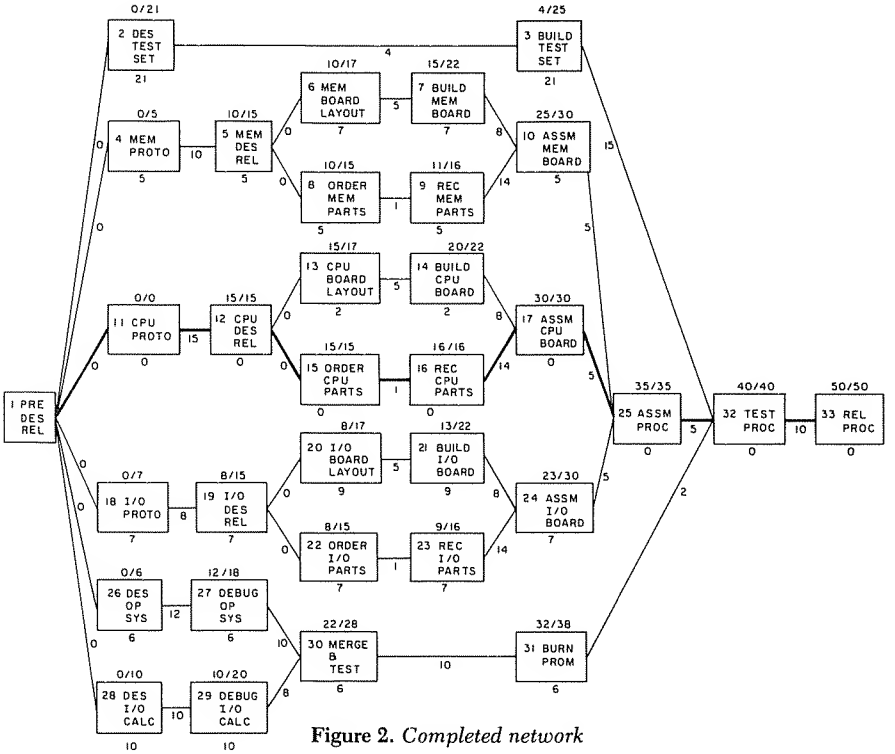


Figure 2. *Completed network*

Referring back to your model, sketch in the major pathways that you know are going to be needed. There is a design phase, a build phase, and a test phase. You know there will be hardware and software efforts. Both hardware and software can be broken down into smaller portions. These portions can be further broken down into easily estimated individual tasks.

Once you have an idea of the overall tasks and the flow of work, you should set down in the table each activity that is involved. If you think of something else that you know will take time or that affects the project, you should put it into the table and find a home for it in your network sketch. When you are satisfied that you have everything under control, you can start to compile the data and assign it to the table and the network. I like to work with the network first.

Place all the names of the activities in the appropriate blocks, in the order in which they will be accomplished. Where certain events require preceding events to be completed first, make sure that the arrows show this. For example, an assembly operation obviously cannot occur before a circuit board is complete and the parts are received. Therefore, the paths must join before any assembly operation can begin. Board layout and an ordering activity, however, can begin as soon as a design prototype is complete. The arrows diverge from the preceding block.

Proceed through the entire network in this manner. When all of the activities have been assigned to a block, you can begin numbering them. Start by numbering each block from the left to the right and from the top to the bottom. Where several arrows merge, do not number that block until all the paths that enter it have been numbered. Where paths diverge, continue to number the uppermost paths first.

When all the numbers have been completed, you can fill in the data table. List all the activities by name in numerical order. Referring to the network, fill in the start and stop events for each activity. The activity name is always the start block, and the termination of the activity is always the start of the next activity. List these start/stop numbers after each activity on the table. Notice that the total number of activities is not the same as the number of start/stop times.

For each activity, you must have a duration, or time in which to complete that activity. Observe that when an activity branches out into a number of new activities, it is assigned zero time. There could be a time assigned to these branching activities, but usually this is just a decision to start, hence no time elapses. If there is a delay before the start of the next activity, then it must, of course, have a duration assigned to it. Using your best information, you can now assign durations to all of the activities in the list and add them to your network beneath each appropriate arrow.

The network now links all of the activities in sequential order. These steps provide the data input for the PERT program. The computer uses this raw

data to develop the early and late times and to search out the critical path.

Once the critical path is revealed, the clever manager can investigate methods of changing the activity duration times by rearranging resources (such as rescheduling people and machines or using premium time) to make sure the project stays on track. He or she can also specify the total project time. The program is set up to make these schedule adjustments easily. Let's see now how the computer uses this data and what kind of results you can hope to get.

The program (see Program Listing) is about 7K long, and when run, uses about 10.5K. It can easily be run with as many as 100 activities on a Level II 16K machine. It will also run on a Model III. The variables used in the program and their definitions are shown in Table 2.

E$(5)	labels
E(100)	activity start times
F(100)	activity finish times
D(100)	durations
TE(100)	early completion times
TL(100)	late completion times
S(100)	slack times
S$(100)	critical path tag
SS(100)	rank slack times
SR(30)	numerical slack times
SC(100)	slack order activity numbers
A$,B$,C$	print formats
Y$,N$,U$,O$	string variables
N	number of activities
M	number of paths
I,J,K,A,B,X,X1,L	array and loop counters
SS,SW,DO,TT,LL,JF MP,GK,PK,MC,ID	temporary variables
J1,I2,L1,L2,LN	print format counters

Table 2. *Program variables*

The program first asks for the name of your project. It then asks for the total number of activities. Enter this from the prepared list. Units are also requested; they can be days, hours, or even weeks or months. The actual units don't matter to the computer as long as you are consistent.

Taking the event numbers and the activity durations from your list, enter the start, finish, and duration for each activity as the computer asks for it, until all the activities have been entered. If the event times are put into the computer out of sequence, the program asks you to reenter the data for that activity. The input routine is located between lines 100 and 210. When all

the data is entered, you are asked if you want to make a data tape. It is not necessary at this time; the program, upon completion of the calculations, reverts to a menu where this option is presented again.

The PERT computations consist of an early event time and a late event time. Both are obtained by accumulating the durations of each activity along their respective branches. All the computations are done in the two subroutines in the 800 and 900 blocks. The early time is the sum in the forward direction; that is, from the front of the network to the back. It is expressed as $TE = \sum_{t0}^{ts} d_e$ where d_e is the duration of each activity. The late time is $TL = \sum_{ts}^{t0} - d_e$, the accumulated subtraction of all of the durations from the end of the network to the beginning.

When the branches merge in the forward direction, the highest TE is used to continue. When the TLs are computed in the backward direction, the lowest TL is used when the branches merge.

The float or slack time is computed in the subroutine from 1000 to 1290. It is the difference TL – TE at each event time. It can be positive or negative, depending on whether you run out of time. The most negative (or least positive) value is the critical path, that path which will be the roadblock to the successful completion of the project.

Once these paths are identified, you can study the network for ways to eliminate the roadblocks and shorten the overall time. Many times, resources that are assigned to less critical activities can be reallocated along the critical path and possibly result in a significant reduction of the project time. Conversely, imagine your embarrassment (to say nothing of your cost) if you were to mistakenly assign critical resources to a path that actually had plenty of slop in it. Believe me, it has been done before and done often. Many a manager has gone out of his or her way to schedule costly overtime on a slack operation, while failing to rescue a key activity as the project slipped slowly down the proverbial drain.

The outputs of this program provide the information needed to avoid such costly mistakes and make the correct decisions to ensure success. The first document is the tabular output of all the computations, as shown in Figure 3. It is arranged in activity number order. The early time, TE, and the late time, TL, are followed by the duration and the slack time. If one of these activities lies on a critical path, it is marked by an asterisk. This routine is located in lines 500 through 670. A printer routine is included in lines 3500 to 3830. These times are normally transferred to your network. Write the early and late times above the appropriate event bubble and write the slack time underneath.

The second output, shown in Figure 4, is a display of all the events in a given path. Each path is presented in the order of the most critical first. This is one of the most important documents in the PERT analysis. It tells at a glance which path is the most critical. Using this output, you can trace each

```
* * * PROGRAM EVALUATION AND REVIEW (PERT) * * *

ANALYSIS OF COMPUTER PROJECT

-------------------------------------------------------------
ACTIVITY     EVENT        TE      TL    DURATION      SLACK
   1       1 TO  2       0.0    21.0       0.0       21.0
   2       2 TO  3       4.0    25.0       4.0       21.0
   3       3 TO 32      19.0    40.0      15.0       21.0
   4       1 TO  4       0.0     5.0       0.0        5.0
   5       4 TO  5      10.0    15.0      10.0        5.0
   6       5 TO  6      10.0    17.0       0.0        7.0
   7       6 TO  7      15.0    22.0       5.0        7.0
   8       7 TO 10      23.0    30.0       8.0        7.0
   9       5 TO  8      10.0    15.0       0.0        5.0
  10       8 TO  9      11.0    16.0       1.0        5.0
  11       9 TO 10      25.0    30.0      14.0        5.0
  12      10 TO 25      30.0    35.0       5.0        5.0
  13       1 TO 11       0.0     0.0       0.0        0.0 *
  14      11 TO 12      15.0    15.0      15.0        0.0 *
  15      12 TO 13      15.0    17.0       0.0        2.0
  16      13 TO 14      20.0    22.0       5.0        2.0
  17      14 TO 17      28.0    30.0       8.0        2.0
  18      12 TO 15      15.0    15.0       0.0        0.0 *
  19      15 TO 16      16.0    16.0       1.0        0.0 *
  20      16 TO 17      30.0    30.0      14.0        0.0 *
  21      17 TO 25      35.0    35.0       5.0        0.0 *
  22       1 TO 18       0.0     7.0       0.0        7.0
  23      18 TO 19       8.0    15.0       8.0        7.0
  24      19 TO 20       8.0    17.0       0.0        9.0
  25      20 TO 21      13.0    22.0       5.0        9.0
  26      21 TO 24      21.0    30.0       8.0        9.0
  27      19 TO 22       8.0    15.0       0.0        7.0
  28      22 TO 23       9.0    16.0       1.0        7.0
  29      23 TO 24      23.0    30.0      14.0        7.0
  30      24 TO 25      28.0    35.0       5.0        7.0
  31       1 TO 26       0.0     6.0       0.0        6.0
  32      26 TO 27      12.0    18.0      12.0        6.0
  33      27 TO 30      22.0    28.0      10.0        6.0
  34       1 TO 28       0.0    10.0       0.0       10.0
  35      28 TO 29      10.0    20.0      10.0       10.0
  36      29 TO 30      18.0    28.0       8.0       10.0
  37      30 TO 31      32.0    38.0      10.0        6.0
  38      31 TO 32      34.0    40.0       2.0        6.0
  39      25 TO 32      40.0    40.0       5.0        0.0 *
  40      32 TO 33      50.0    50.0      10.0        0.0 *

THE CRITICAL PATH LENGTH IS   50.0 DAYS
```

Figure 3

path on the network diagram and label all of the appropriate slack times.

Paths with identical slack times occasionally occur. These may be missed by the program. A small change in their length that makes them unequal forces them to be identified. The change routine in the program can be used to do this.

The third document is the activity/time plot, shown in Figure 5. A time scale is printed across the top of the page. Its length is from the beginning of the project to the last event and is scaled to occupy the top of the screen or printer. The computations for this segment are done in the subroutine in lines 6000 to 6990.

```
PATHS IN CRITICAL ORDER FOR COMPUTER PROJECT ARE :

PATH  1 :  ------------SLACK=>  0 DAYS
 1 - 11 - 12 - 15 - 16 - 17 - 25 - 32 - 33
PATH  2 :  ------------SLACK=>  2 DAYS
 12 - 13 - 14 - 17
PATH  3 :  ------------SLACK=>  5 DAYS
 1 - 4 - 5 - 8 - 9 - 10 - 25
PATH  4 :  ------------SLACK=>  6 DAYS
 1 - 26 - 27 - 30 - 31 - 32
PATH  5 :  ------------SLACK=>  7 DAYS
 5 - 6 - 7 - 10 - 18 - 19 - 22 - 23 - 24 - 25
PATH  6 :  ------------SLACK=>  9 DAYS
 19 - 20 - 21 - 24
PATH  7 :  ------------SLACK=>  10 DAYS
 1 - 28 - 29 - 30
PATH  8 :  ------------SLACK=>  21 DAYS
 1 - 2 - 3 - 32
```

Figure 4

```
          <<<<<<<<<<ACTIVITY-TIME PLOT>>>>>>>>>>

   PROJECT COMPUTER PROJECT

  0   3   7  10  13  17  20  23  27  30  33  37  40  43  47  50
-------------------------------------------------------------------

I
I----  1 11------ 12 15 16-------- 17--- 25-- 32--------- 33
I
I--------------- 12 13-- 14------ 17!        + 2D
I
I----  1  4------- 5  8  9---------- 10--- 25!+ 5D
I
I----  1 26---------- 27-------- 30-------- 31 32! + 6D
I
I--------------  5-  6---  7------- 10 18 19 22 23 24 25!+ 7D
I
I--------------- 19 20-- 21------ 24!        + 9D
I
I----  1---- 28-------- 29------ 30!        +10D
I
I----  1---------------  2---  3-------------- 32!+21D
```

Figure 5

The events of each path appear in a row beneath the scale. The last event in each path is terminated with an exclamation point. If you were to extend a line from this last event upward to the time scale, you could read off the approximate time of completion of that path. If the first entry of that path is dependent upon the completion of another activity, then it appears at its appropriate start time, and a line drawn upward to the time scale gives the approximate start time. All of these times can be assigned calendar days or weeks consistent with the units chosen. The critical path, of course, consumes all of the time and is the end of the scale.

All earlier paths have slack, and the plot displays the amount of slack in the space following the last event in that path. Projects that consist of large numbers of paths or branches are paged on the screen. At the end of each of

the three outputs the program can be returned to the menu where several more options are presented. Two of these options allow the program to be recomputed with different times. The first permits a change in any given activity duration. This would occur if you wished to reallocate either people or machines from a slack path to the critical path. The path would shorten (have less slack), and the critical path would then pick up the slack. A new path might then become critical, and the total project time would change, for the better, you hope.

With the second option, it is possible to change and thus specify the total project time. If this time is less than possible, the critical paths become negative. The project manager now faces the task of allocating resources until all of the negative paths are cleared and the project time is molded to fit the required time. This changed data can be stored on tape for future use through the tape output and input routines. These routines are located at lines 4000 and 5000 in the program. They can be selected from the menu as needed.

I have used this program in actual business situations. The information obtained was impressive and it was easy to sell others on the need for reallocation of valuable resources. With these three documents, the manager of the small business has a powerful tool with which to plan a project and make intelligent decisions based on a knowledgeable appreciation of the impact of a variety of tasks and their complex interaction on the project as a whole. Using your microcomputer, you can now finish the job with the least cost and finish it on time.

Program Listing. *PERT*

Encyclopedia Loader

```
10 REM PROGRAM EVALUATION AND REVIEW TECHNIQUE (PERT)
12 REM JAMES DEVLIN,LAKEVIEW,N.Y.14085  DEC.1981
15 FOR I=1 TO 3:OUT 1,0:NEXT:OUT 1,64:OUT 1,250:OUT 1,51
20 CLEAR 200:DEFINT E,F,I,J,M,K,X
30 DIM E(100),F(100),D(100),TE(100),TL(100),M1(30),M2(30)
40 DIM S(100),SR(30),E$(5),S$(100),SS(100),SC(100)
50 TT=0:A$="####.#":B$="###":C$="##"
60 FOR I=1 TO 5:READ E$(I):NEXT
70 CLS:PRINT:PRINT"* * * P E R T   A N A L Y S I S * * *"
80 Y$="N":INPUT"DATA FILE FROM TAPE: (Y/N)";Y$
90 IF Y$="Y" THEN 5000
100 REM DATA INPUT ROUTINES
105 INPUT"ENTER NAME OF PROJECT";N$
110 PRINT:INPUT"ENTER THE TOTAL NUMBER OF ACTIVITIES";N
120 INPUT"WHAT UNITS WILL BE USED...DAYS/HRS";U$
130 INPUT"IS THERE A SPECIFIED COMPLETION TIME (Y/N)";Y$
140 IF Y$="Y" THEN INPUT"ENTER TOTAL DURATION";TD:TT=1
150 PRINT:FOR I=1 TO N:PRINT E$(1);I
170 PRINT E$(2);:INPUT E(I):PRINT E$(3);:INPUT F(I)
180 IF F(I)<=E(I) OR F(I)>N THEN PRINT"SEQUENCE ERROR":GOTO 170
200 PRINTE$(4);:INPUTD(I)
210 NEXTI
300 Y$="N":INPUT"STORE THIS DATA ON TAPE-Y/N";Y$
310 IF Y$="Y" THEN GOSUB 4000
320 GOSUB 800
330 GOSUB 900
340 GOSUB 1000
470 CLS:PRINT:PRINT"WOULD YOU LIKE :":PRINT
475 O$="NONE":PRINT"1. FULL OUTPUT TABLE":PRINT"2. CRITICAL PATHS"
480 PRINT"3. ACTIVITY-TIME PLOT":PRINT"4. CHANGE ACTIVITY TIMES"
485 PRINT"5. CHANGE PROJECT COMPLETION TIME":PRINT"6. WRITE TAPE"
490 INPUT"ENTER OPTION DESIRED";X1:IF X1<1 OR X1>6 THEN 470
495 ON X1 GOSUB 500,1300,6000,2300,2400,4000:GOTO 470
500 REM PRINT ROUTINE
510 INPUT"FOR HARDCOPY PRINTOUT,TYPE 'TTY'";O$
520 IF O$="TTY" GOSUB 3500
530 GOSUB 2500
540 K=10:FOR I=1 TO N:K=K-1
550 PRINTTAB(2)I;TAB(9)E(I);" TO ";F(I);
560 TE=TE(E(I))+D(I):TL=TL(F(I))
570 PRINTTAB(21)USINGA$;TE;:PRINTTAB(29)USINGA$;TL;
580 PRINTTAB(39)USINGA$;D(I);
590 PRINTTAB(50)USINGA$;S(I);:PRINTS$(I)
595 IF O$="TTY" GOSUB 3600
600 IF K <= 0 THEN INPUT"HIT ENTER TO CONTINUE";X:K=10:GOSUB 2500
610 NEXT I
620 PRINT:PRINT "THE CRITICAL PATH LENGTH IS ";
630 PRINTUSINGA$;TL(F(N));:PRINT" ";U$
650 IF O$="TTY" GOSUB 3700
660 INPUT"HIT INPUT FOR OPTIONS";X
670 RETURN
800 REM SUM FORWARDS -(TE(I))
810 FOR I=1 TO N:TE(I)=0:TL(I)=0:NEXT I
820 PRINT "* CALCULATING *":FOR I=1 TO N
830 IF TE(F(I))>=TE(E(I))+D(I) THEN 860
840 TE(F(I))=TE(E(I))+D(I)
860 NEXT I
870 IF TT>0 THEN TL(F(N))=TD ELSE TL(F(N))=TE(F(N))
880 NE=TE(F(N))
890 RETURN
900 REM SUM REVERSE FOR TL(I)
910 FOR I=N TO 1 STEP-1
920 IF TL(E(I))=0 OR TL(E(I))>TL(F(I))-D(I) THEN 940
930 GOTO 960
940 TL(E(I))=TL(F(I))-D(I)
960 S(I)=TL(F(I))-(TE(E(I))+D(I))
970 IF S(I)<=0 THEN S$(I)=" *" ELSE S$(I)=""
```

```
980 NEXT I
990 RETURN
1000 REM SLACK SORT
1020 FOR I=1 TO N:SS(I)=S(I):NEXT I
1040 SW=0
1050 FOR I=1 TO N-1
1060 IF SS(I)<=SS(I+1) THEN 1110
1070 SS=SS(I)
1080 SS(I)=SS(I+1)
1090 SS(I+1)=SS
1100 SW=1
1110 NEXT I
1120 IF SW=1 THEN 1040
1200 SS=SS(1):J=1:SR(J)=SS:REM DELETE REDUNDANT ROWS
1210 FOR I=1 TO N
1220 IF SS(I)<=SS THEN 1240
1230 J=J+1:SR(J)=SS(I):SS=SS(I)
1240 NEXT I:M=J
1250 K=1:FOR I=1 TO M
1260 M1(I)=K:FOR J=1 TO N
1270 IF S(J)=SR(I) THEN SC(K)=J:M2(I)=K:K=K+1
1280 NEXT J:NEXT I
1290 RETURN
1300 REM PRINT PATHS
1305 INPUT"FOR HARDCOPY,TYPE 'TTY'";O$
1310 CLS:PRINT:PRINT"PATHS IN 'CRITICAL' ORDER FOR ";N$;" ARE:""
1315 IF O$="TTY" GOSUB 3800
1320 K=5:FOR I=1 TO M:K=K-1
1330 PRINT:PRINT"PATH ";I;":  ----------";E$(5);"=> ";SR(I);U$
1340 IF O$="TTY" GOSUB 3820
1350 L1=0:L2=0:L=M1(I)
1360 GOSUB 7400
1370 PRINT TF;:IF O$="TTY" THEN LPRINT TF;
1380 IF TF>=F(SC(M2(I))) THEN 1410
1390 PRINT"-";:IF O$="TTY" THEN LPRINT"-";
1400 GOTO 1360
1410 K1=K1+1
1420 IF K<=0 THEN PRINT:INPUT"HIT ENTER TO CONTINUE";X:CLS:K=5
1430 NEXT I
1480 PRINT:IF O$="TTY" THEN LPRINT
1490 INPUT"HIT ENTER FOR OPTIONS";X:RETURN
2300 Y$="N":INPUT"SELECT ACTIVITY FOR CHANGE";I
2310 PRINT E$(1);I;"    ";E$(2);E(I);"    ";E$(3);F(I)
2320 PRINT"OLD ";E$(4);" IS ";D(I):DO=D(I)
2330 PRINT"NEW ";E$(4);:INPUT" WILL BE ";D(I)
2370 PRINT:INPUT"ANOTHER CHANGE  (Y/N)";Y$
2380 IF Y$="Y" THEN 2300 ELSE IF D(I)<>DO THEN 320 ELSE 470
2400 CLS:T2=TL(F(N)):PRINT"THE LATEST POSSIBLE DATE FOR PROJECT"
2410 PRINT"COMPLETION IS :";TL(F(N));U$
2420 PRINT:INPUT"DESIRED TIME IS ";TD:TT=1
2430 IF TD<>T2 THEN 320 ELSE RETURN
2500 CLS:PRINT"ANALYSIS OF ";N$:PRINT STRING$(54,"-")
2510 PRINT E$(1);TAB(12)"EVENT";
2520 PRINTTAB(24)"TE";TAB(32)"TL";
2530 PRINTTAB(37)E$(4);
2540 PRINTTAB(52)E$(5)
2550 RETURN
3500 REM TTY OUTPUT ROUTINES
3510 LPRINT"* * * PROGRAM EVALUATION AND REVIEW (PERT) * * *"
3520 LPRINT:LPRINT"ANALYSIS OF ";N$:LPRINT
3530 LPRINT STRING$(54,"-")
3540 LPRINT"ACTIVITY";TAB(12)"EVENT";
3550 LPRINTTAB(24)"TE";TAB(32)"TL";
3560 LPRINTTAB(37)"DURATION";
3570 LPRINTTAB(52)"SLACK"
3580 RETURN
3600 REM TTY PRINT DATA
3610 LPRINTTAB(2)I;TAB(9)E(I);" TO ";F(I);
3620 LPRINTTAB(21)USINGA$;TE;:LPRINTTAB(29)USINGA$;TL;
3630 LPRINTTAB(39)USINGA$;D(I);
```

Program continued

```
3640 LPRINTTAB(50)USINGA$;S(I);:LPRINTS$(I)
3650 RETURN
3700 LPRINT:LPRINT "THE CRITICAL PATH LENGTH IS ";
3710 LPRINTUSINGA$;TL(F(N));:LPRINT" ";U$
3730 RETURN
3800 LPRINT:LPRINT"PATHS IN CRITICAL ORDER ";
3810 LPRINT"FOR ";N$;" ARE :":RETURN
3820 LPRINT:LPRINT"PATH ";I;":  ----------";E$(5);"=> ";SR(I);U$
3830 RETURN
4000 PRINT"PUT DATA FILE TAPE IN RECORDER"
4010 INPUT"ARE RECORD & PLAY SET (Y/N)";Y$
4020 IF Y$<>"Y" THEN 4000
4030 PRINT@650,"RECORDING DATA FILE ";N$
4040 PRINT#-1,N$,N,U$
4050 FOR I=1 TO N
4060 PRINT#-1,E(I),F(I),D(I)
4070 NEXT I
4080 RETURN
5000 PRINT"PLACE DESIRED TAPE FILE IN RECORDER"
5010 INPUT"ENTER FILE NAME";NN$
5020 INPUT"HIT ENTER WHEN READY";X
5030 INPUT#-1,N$,N,U$
5040 IF N$<>NN$ PRINT "WRONG FILE":GOTO 5000
5050 PRINT@650,"READING FILE ";N$
5060 FOR I=1 TO N
5070 INPUT#-1,E(I),F(I),D(I)
5080 NEXT I
5090 GOTO 320
6000 REM ACTIVITY-TIME DIAGRAM
6010 INPUT"FOR HARDCOPY TYPE 'TTY'";O$:O1=0
6100 MP=M/6:GK=NE/15:PK=56/NE
6110 GOSUB 7200
6150 MC=0:FOR I=1 TO M
6160 FT=F(SC(M2(I)))
6170 PRINT:PRINT"I":PRINT"I";
6180 IF O$="TTY" THEN LPRINT:LPRINT"I":LPRINT"I";
6190 JF=0:L1=0:L2=0:L=M1(I)
6200 FOR J=1 TO 60
6210 IF JF=1 THEN 6230
6220 GOSUB 7400
6230 J1=TL(TF)*PK
6240 IF J>=J1 THEN PRINTUSINGB$;TF;:J2=J1:JF=0:GOTO 6260
6245 IF O$="TTY" THEN LPRINTTAB(J);"-";
6250 PRINTTAB(J);"-";:GOTO 6280
6260 IF O$="TTY" THEN LPRINTUSINGB$;TF;
6265 J=J+LN-1
6270 IF TF>=FT THEN GOSUB 7300:GOTO 6290
6280 NEXT J
6290 M2=M2+1
6300 IF M2<=6 THEN 6350
6310 IF MC>=MP THEN 6990
6320 PRINT@1016,"ENTER";:INPUTX:MC=MC+1:GOSUB 7200
6350 NEXT I
6990 PRINT@1016,"ENTER";:INPUTX:GOTO 470
7200 CLS:M2=1:FOR K=0TO15:PRINTTAB(K*4)USINGB$;(K*GK);:NEXT K
7210 PRINT:PRINT STRING$(63,"-");
7230 IF O$="TTY" GOSUB 7500
7240 RETURN
7300 ID=58-J1:I2=ID/2:IF SR(I)<1 OR ID<4 THEN 7340
7310 PRINT"!";:PRINTTAB(J1+I2);"+";
7315 PRINTUSINGC$;SR(I);:PRINTLEFT$(U$,1);
7320 IF O$<>"TTY" THEN 7340
7325 LPRINT"!";:LPRINTTAB(J1+I2)"+";
7330 LPRINTUSINGC$;SR(I);:LPRINTLEFT$(U$,1);
7340 RETURN
7400 REM ACTIVITY # SUB
7410 IF L1=1 AND L2=1 THEN TF=F(SC(L)):GOTO 7450
7420 IF L2=1 AND L1=0 THEN TF=F(SC(L)):L1=1:GOTO 7450
7440 TF=E(SC(L)):L2=1:GOTO 7470
7450 L=L+1
```

```
7470 TF$=STR$(TF):LN=LEN(TF$):JF=1:RETURN
7500 REM ACTIVITY-TIME PRINT OUT
7510 IF O1=1 THEN 7600
7520 O1=1
7540 LPRINT:LPRINTTAB(10)STRING$(10,"<");
7550 LPRINT"ACTIVITY-TIME PLOT";:LPRINTSTRING$(10,">")
7560 LPRINT:LPRINT"    PROJECT ";N$
7570 LPRINT:LPRINT
7580 FOR K=0 TO 15:LPRINTTAB(K*4)USINGB$;(K*GK);:NEXT K:LPRINT
7590 LPRINTSTRING$(63,"-")
7600 RETURN
8000 DATA ACTIVITY,START,FINISH,DURATION,SLACK
9999 END
```

EDUCATION

Physics in Motion—
Exploring the Projectile Problem

Physics in Motion—Exploring the Projectile Problem

by Linda Huetinck and Debra Lelewer

Computers are used in education both to perform administrative tasks and to increase students' interest in such subjects as math and spelling by providing drill. The immediate feedback provided by a computer experience along with the fascination a student feels for the machine itself can be a strong motivational tool. The ultimate computer exercise, however, is one in which the student learns new concepts rather than simply practicing old ones. We've written the Space Shuttle Pilot (Program Listing 1) and Human Cannonball (Program Listing 2) programs to help students understand the physics/mathematics concepts of projectile motion.

To enhance the student's interest in the programs, we used a little imagination in creating exciting tasks. In the first program, the student is a space shuttle pilot making a practice landing; in the second program he or she is training to be a human cannonball in a circus. The first program deals with the simplified equations of motion that are a result of firing a projectile horizontally (that is, angle θ is zero). As a human cannonball fired up at an angle θ, the student must use the more generalized equations that include sin θ and cos θ.

Operation

The first exercise in the space shuttle program emphasizes the importance of taking projectile motion apart into horizontal and vertical components. The computer displays the path of a ball tossed horizontally, using graphics blocks to form a parabolic path. The student graphs the x- and y-coordinates separately to observe the components of motion. The computer then supplies the general equations for horizontal distance (x) and vertical distance (y) in terms of time (t). Given a distance y, the student uses these equations to calculate t and x. The student is then asked to apply the equations to the space shuttle.

The cannonball program gives the angle and the speed at which the student will be fired from a cannon. The student is then asked to calculate vertical and horizontal components of velocity using basic trigonometric functions. When he or she has done this correctly, the program supplies equations governing motion as a projectile (distance and velocity equations in both the horizontal and vertical directions). The student sees the path of the projectile in two ways. First, a parabolic path formed by graphics blocks appears on the screen. Then, the same path is shown a second time, with the

graphics blocks appearing slowly, one after another. This time, the student is instructed to stop the projectile motion at any point. When he or she stops the projectile, the horizontal distance for that point is given. The student is asked to calculate the vertical distance and the velocity components at this time.

As a final exercise, the problem is expanded so that the student is finding final and maximum distances. We give the height and width of the circus tent and ask whether the human cannonball will hit the top of the tent and how far away a net should be placed so that it catches him or her. The answers to these questions would be submitted to the teacher as the lab write-up.

Techniques

These programs are written in TRS-80 Level II Disk BASIC. Extensive use of the INKEY$ function allows the student to move through the program at his or her own pace. After every set of instructions, the program loops, waiting for the student to enter a C to indicate that he or she is ready to go on. A loop was also used to slow down the graphics display of the parabola the second time it occurs. The subroutine contains a timing loop and an INKEY$ function which allows the student to stop the motion of the projectile. Early in the program, when the student makes a sample calculation, the answers are checked. If the student's answer is within two-tenths of the actual answer, the computer gives praise and instructions to continue. When the answer is incorrect, the computer response gives a hint as to the problem and then gives the student the option of trying again or terminating the program and seeing the teacher for help.

The programs are short, each requiring only 20 to 30 minutes. To save time, not all answers are checked by the computer. Since these machines are in the back of a classroom that might be disturbed by printer noise, we have the students copy the data from the video screen.

The reaction of our students to the Space Shuttle Pilot and Human Cannonball programs has been very positive. The humor and visual aspect of this lab experience combine to make a difficult concept less threatening. The students showed much better understanding of these concepts after going through these programs than was demonstrated in previous years after the typical textbook approach. We plan to expand the cannonball program for use with more advanced calculus students. These students could derive the equations in lines 290–300 and enter them as input, rather than having the program supply these equations.

Program Listing 1. *Space Shuttle*

Encyclopedia Loader

```
5 REM PROJECTILE PILOT TRAINING BY HUETINCK/LELEWER (PROJPT/BAS)
10 CLS
20 PRINT"WELCOME TO THE PHASCINATING WORLD OF PHYSICS."
30 PRINT
40 PRINT"TODAY IS THE FIRST DAY OF YOUR PHYSICS PHLIGHT TRAINING"
50 PRINT"TO BECOME A SPACE SHUTTLE PILOT FOR NASA."
60 PRINT"THE RE-ENTERING VECHICLE SHUTS OFF ITS ENGINES AND BECOMES"
70 PRINT"A FREE FALL PROJECTILE, SO FIRST ALL ABOUT PROJECTILES--"
80 PRINT
90 PRINT"TO START THE STUDY OF PROJECTILES, TOSS A BALL HORIZONTALLY"
100 PRINT"AT 5 M/SEC AND TAKE A STROBE PICTURE."
110 PRINT"I WILL INDICATE THE COORDINATES. COPY THE DATA AND GRAPH IT.
    "
130 PRINT
140 PRINT"PRESS THE LETTER 'C' WHEN YOU ARE READY TO CONTINUE."
145 PRINT"AFTER YOU ARE THROUGH COPYING THE DATA, PRESS 'C' AGAIN."
150 IFINKEY$="C" THEN GOTO 160 ELSE 150
160 CLS
170 PRINT@34,"COORDINATES OF POSITIONS SHOWN"
180 PRINT@181,"(0,10)":PRINT@245,"(.7,9.9)":PRINT@309,"(1.4,9.6)"
190 PRINT@373,"(2.1,9.1)":PRINT@437,"(2.8,8.4)":PRINT@501,"(3.5,7.5)"
200 PRINT@565,"(4.2,6.4)":PRINT@629,"(4.9,5.1)":PRINT@693,"(5.6,3.6)"
210 PRINT@757,"(6.3,1.9)":PRINT@821,"(7,0)"
220 SET (0,1):SET(5,2):SET(10,4):SET(15,7)
230 SET(20,11):SET(25,16):SET(30,22):SET(35,29)
240 SET (40,37):SET(45,45)
250 IFINKEY$="C" THEN GOTO 255 ELSE 250
255 CLS
260 PRINT"NOW GRAPH THE HORIZONTAL COMPONENTS ONLY, ON A HORIZONTAL LI
    NE."
270 PRINT"(LET Y=0 AND PLOT THE X VALUES)"
280 PRINT"I'LL WAIT UNTIL YOU ARE THROUGH. PRESS 'C' TO CONTINUE"
285 IFINKEY$="C" THEN GOTO 290 ELSE 285
290 PRINT
300 PRINT"THEY ARE EVENLY SPACED!!! JUST LIKE A HORIZONTAL DRY ICE PUC
    K"
310 PRINT"THE VELOCITY IS CONSTANT AND THE ACCELERATION IS 0."
315 PRINT
320 PRINT"NOW GRAPH THE VERTICAL COMPONENTS ONLY ON A VERTICAL LINE"
330 PRINT"(LET X=0 AND PLOT THE Y VALUES)."
340 PRINT"I'LL WAIT UNTIL YOU ARE THROUGH. PRESS 'C' TO CONTINUE."
345 IFINKEY$="C" THEN GOTO 348 ELSE 345
348 PRINT
350 PRINT"THEY ARE INCREASING REGULARLY!! JUST LIKE A FREELY FALLING B
    ODY."
360 PRINT"THE VELOCITY IS 0 AND THE ACCELERATION IS G."
365 PRINT"PRESS 'C' TO CONTINUE"
368 IFINKEY$="C" THEN GOTO 370 ELSE 368
370 CLS
380 PRINT"THE GENERAL DISTANCE EQUATION IS  D = VI*T + 1/2 * A*T[2"
390 PRINT
400 PRINT "IN THE X DIRECTION, VI = 5, A = 0 WHICH GIVES X = VI*T"
410 PRINT"REMEMBER THE GRAVITY IS ONLY ACCELERATING THE OBJECT DOWN."
450 PRINT
460 PRINT"IN THE Y DIRECTION, VI = 0, A = G WHICH GIVES Y = 1/2*G*T[2"

470 PRINT"REMEMBER THE VI = 0 BECAUSE IT WAS FIRED HORIZONTALLY."
480 PRINT
490 PRINT"THE POINT IS THAT WE HAVE TAKEN THE MOTION APART INTO ITS "
500 PRINT"HORIZONTAL AND VERTICAL COMPONENTS."
505 PRINT"TYPE 'C' WHEN READY TO CONTINUE."
510 IFINKEY$= "C" THEN 515 ELSE 510
515 CLS
520 PRINT"NOW USING THESE TWO EQUATIONS X = VI*T AND Y=1/2*A*T[2"
525 PRINT"SOLVE THIS PROBLEM:"
530 PRINT"A CANNONBALL IS FIRED HORIZONTALLY WITH VELOCITY OF 88 M/SEC
    "
540 PRINT"FROM THE TOP OF A CLIFF 56 M HIGH."
```

Program continued

```
550 INPUT "IN WHAT TIME WILL IT STRIKE THE PLAIN AT THE CLIFF'S BASE";
    T
560 IF T >3.2 AND T < 3.5 THEN 590 ELSE 570
570 PRINT"TRY AGAIN. DID YOU USE A = 9.8 M/SEC[2 ?"
580 GOTO 550
590 PRINT"YOU DID IT!!!"
600 INPUT "AT WHAT DISTANCE X FROM THE FOOT OF THE CLIFF WILL IT STRIK
    E";X
610 IF X>289 AND X<300 THEN 640 ELSE 620
620 PRINT"TRY AGAIN, IT'S NOT THAT HARD."
630 GOTO 600
635 PRINT
640 PRINT"NOW YOU'RE READY FOR THE FINAL EXAM. TYPE 'C' WHEN READY."
650 IFINKEY$="C" THEN GOTO 660 ELSE 650
660 CLS
670 PRINT"THE SPACE SHUTTLE IS ON THE 747 BOMBER FOR A TEST LANDING."
675 PRINT"IT WILL NOT USE ITS MOTORS BUT COME IN BY FREE FALL."
690 PRINT"THE SHUTTLE CARRIER AIRCRAFT IS FLYING HORIZONTALLY"
700 PRINT"1200 M ABOVE THE GROUND AT 400 M/SEC."
705 PRINT"HOW FAR HORIZONTALLY FROM THE TARGET SHOULD THE PILOT SEPARA
    TE"
707 PRINT"THE SPACE SHUTTLE TO HAVE IT COME IN RIGHT IN FRONT OF THE"
708 PRINT"BULL'S EYE TV CAMERAS?"
730 PRINT
736 PRINT"TURN THIS PROBLEM IN WITH YOUR LAB."
738 PRINT
740 PRINT"GOOD LUCK. I HOPE YOU GET YOUR FLIGHT WINGS FOR OUTER SPACE.
    "
750 PRINT"COME BACK AND SEE ME AGAIN."
760 END
```

Program Listing 2. *Human Cannonball*

```
10 CLS:REM  PROJECTILE MOTION PROGRAM AT AN ANGLE BY HUETINCK/LELEWER
    (CANNON/BAS)
20 PRINT"WELCOME BACK TO THE PHASCINATING WORLD OF PHYSICS."
30 PRINT"YOU HAVE LANDED A SUMMER JOB WITH THE RINGLING BROTHERS CIRCU
    S,"
40 PRINT"WHERE YOU WILL BECOME A PHYSICS PHENOMENON."
50 PRINT"YOUR ASSIGNMENT IS TO BE A HUMAN CANNONBALL IN THE"
55 PRINT"          ** GRANDE FINALE **":PRINT:PRINT:PRINT
70 PRINT"BEFORE REPORTING TO SARASOTA, FLORIDA FOR TRAINING"
80 PRINT"YOU SHOULD OVERCOME YOUR FEAR OF FLYING"
90 PRINT"BY LEARNING A LITTLE ABOUT PROJECTILE MOTION.":PRINT
100 PRINT"PRESS 'C' WHEN YOU'RE READY FOR YOUR LESSON."
110 IF INKEY$="C" THEN 120 ELSE 110
120 CLS:PRINT"YOU WILL BE FIRED FROM THE CANNON"
140 PRINT"AT AN ANGLE OF 50 DEGREES AND WITH A VELOCITY OF 25 M/S."
150 PRINT@768,"USE BASIC TRIG FUNCTION DEFINITIONS TO SOLVE FOR VX AND
    VY."
172 PRINT@339," 25":PRINT@535,"50":PRINT@665,"VX":PRINT@360,"VY"
180 FOR I=9 TO 27:SET(2*(59-I)-30,I):NEXT I
186 FOR J=34 TO 70:SET(J,27):NEXT J
192 FOR J=9 TO 27 STEP 2:SET(70,J):NEXT J
198 PRINT@903,"VX";:INPUT VX
200 PRINT@967,"VY";:INPUT VY
210 IF VX>=15.9 AND VX<=16.2 THEN K=1 ELSE K=2
220 IF VY>=18.9 AND VY<=19.3 THEN N=1 ELSE N=2
230 IF K=1 AND N=1 THEN 245
232 GOTO 320
245 CLS:PRINT"RIGHT!  YOU'VE PASSED THE AUDITION."
260 PRINT"NOW USE THESE EQUATIONS TO CONTINUE TRAINING."
270 PRINT"WRITE DOWN THE EQUATIONS SO THAT YOU'LL HAVE THEM.":PRINT:PR
    INT
290 PRINT"VX = 16.07                VY =  -9.8*T + 19.15      ":PRINT
300 PRINT"X  = 16.07*T              Y  = -4.9*T[2 + 19.15*T
    "
```

```
310 PRINT:PRINT:PRINT"WHEN YOU HAVE THE EQUATIONS, TYPE 'C' TO CONTINU
    E."
315 IF INKEY$="C" THEN 400 ELSE 315
320 CLS:PRINT"WRONG -- NO CIGAR!"
360 PRINT"TRY AGAIN - GET HELP - DON'T LOSE OUT ON THIS"
370 PRINT"GOLDEN OPPORTUNITY FOR A HIGH-PAYING JOB"
380 PRINT"WITH A CHANCE TO SEE THE WORLD!":PRINT:PRINT
384 IF K=1 AND N=2 PRINT"VX WAS RIGHT, BUT VY WAS WRONG."
386 IF K=2 AND N=1 PRINT"VY WAS RIGHT, BUT VX WAS WRONG."
388 IF K=2 AND N=2 PRINT"VX AND VY WERE BOTH WRONG."
390 INPUT"ARE YOU READY TO TRY AGAIN (Y OR N)";A$
392 IF A$="Y" THEN 120
394 IF A$="N" PRINT"COME BACK AFTER YOU GET HELP FROM YOUR TEACHER.":E
    ND
396 GOTO 390
400 CLS:PRINT"HERE'S A PREVIEW OF THE PATH YOU'LL TAKE"
420 PRINT"AS YOU'RE FIRED FROM THE CANNON."
430 SET(32,33):SET(39,27):SET(46,22):SET(53,18):SET(60,15):SET(67,13):
    SET(74,12)
440 SET(81,13):SET(88,15):SET(95,18):SET(102,22):SET(109,27):SET(116,3
    3)
500 PRINT@832,"WE'LL LOOK AT THE PATH IN A DIFFERENT WAY.  TYPE 'C' WH
    EN READY."
510 PRINT@896,"THIS TIME, STOP YOUR MOTION AT ANY POINT BY TYPING 'S'.
    "
515 IF INKEY$="C" THEN 520 ELSE 515
520 CLS
522 SET(32,33):P=0:GOSUB 1000
524 SET(39,27):P=1:GOSUB 1000
526 SET(46,22):P=2:GOSUB 1000
528 SET(53,18):P=3:GOSUB 1000
530 SET(60,15):P=4:GOSUB 1000
532 SET(67,13):P=5:GOSUB 1000
534 SET(74,12):P=6:GOSUB 1000
536 SET(81,13):P=7:GOSUB 1000
538 SET(88,15):P=8:GOSUB 1000
540 SET(95,18):P=9:GOSUB 1000
542 SET(102,22):P=10:GOSUB1000
544 SET(109,27):P=11:GOSUB 1000
546 SET(116,33):P=12:GOSUB 1000
550 GOTO 500
575 T=.326*P:X=16.07*T:PRINT@652,"X = ";:PRINT USING"##.#";X
600 PRINT"NOW WE HAVE SOME QUESTIONS FOR YOU.  WRITE DOWN THE X VALUE"

605 PRINT"AND WRITE DOWN THE FOLLOWING QUESTIONS.  YOU'LL TURN IN"
610 PRINT"THE ANSWERS AT THE END OF YOUR TRAINING."
612 PRINT"          1) WHAT IS THE VALUE OF Y AT THIS POINT?"
614 PRINT"          2) WHAT ARE VX AND VY NOW?":PRINT
620 PRINT"TYPE 'C' WHEN YOU'RE READY TO CONTINUE."
622 IF INKEY$="C" THEN 650 ELSE 622
650 CLS:PRINT"YOU MAY HAVE A FEW ADDITIONAL PRACTICAL CONCERNS"
670 PRINT"ABOUT THIS NEW OCCUPATION OF YOURS.":PRINT:PRINT
680 PRINT"THE TENT IN WHICH YOU PERFORM IS 25 M  HIGH AND 80 M ACROSS.
    "
690 PRINT"IF YOU'RE SHOT ACROSS THE TENT TOWARD A NET,"
710 PRINT@466,"1)  WILL YOU HIT THE TOP OF THE TENT?"
715 PRINT@530,"2)  HOW FAR AWAY SHOULD THE NET BE PLACED?"
720 PRINT@704,"TURN IN YOUR ANSWERS FOR THESE TWO QUESTIONS."
730 PRINT@768,"YOUR TRAINING IS NOW COMPLETE.  GOOD LUCK ON OPENING NI
    GHT!"
750 END
1000 FOR N=1 TO 50:IF INKEY$<>"" THEN 575
1020 NEXT N
1030 RETURN
```

GAMES

Satan's Square
Card Playing
Another Magic Trick

Satan's Square

by James Wood

Satan's Square is the TRS-80's answer to Rubik's Cube™. It contains all the fun and challenge of a real Rubik's Cube, with the exception that you cannot throw it against a wall when the frustration level becomes too great. The game requires a 16K Color Computer with Extended Color BASIC.

When you run the game, you are first asked to enter your skill level, a number from 3 to 8. Each row of the cube will be made up of the number of cells that you enter. For example, if you enter 3, the cube will have three cells per side; if you enter 8, the cube will have eight cells per side. Next, you are asked whether you would like the rows to be differentiated by color or by a different letter in each row. If you select C (for color), the rows of cells will be of different colors, very much like an actual Rubik's Cube. If you answer L (for letter), all cells will be of the same color, but each row will have a different letter in each cell. This option allows you to play the game on a black and white television (or with color-blind players!).

When the game begins, a pattern of cells arranged in a square is displayed. The cells in each row are the same color, or letter, depending on what you selected. You are then asked if you would like the cube to be shuffled. Respond by striking Y if you want the computer to mix the cells, or N if you want to begin the game with an orderly cube. If you select Y (yes, shuffle), the cells will be shuffled first vertically, then horizontally.

Moves are made by typing a number corresponding to the row or column, then an arrow for the direction desired. All the cells will be moved in the direction indicated by the arrow, with the last cell wrapping around to the first position (see Figure 1). The computer keeps track of the number of moves you have made and displays them in the lower right corner of the screen. If the pressure becomes too much for you, hit the Z key. This causes the computer to ask if you wish to play again. Respond with Y for yes, N for no.

How the Program Works

Table 1 lists program lines and descriptions. Lines 4–26 display the instructions and input the information needed to set up the cube. Error traps are provided in lines 20 and 26 to guard against illegal answers.

Lines 28–30 clear the screen to black and draw the cube. Lines 40 and 41 ask if the cube is to be shuffled. If you respond with Y (shuffle), execution progresses to line 50; an N response causes a jump to line 160. Any other

response is an illegal one, causing a loop back to the INKEY$ function in line 41. Lines 51–85 make up the vertical shuffle, and lines 99–150 comprise the horizontal shuffle. Lines 165–171 maintain a count of the moves in variable X and display the number of moves on the screen.

Line 190 is a common INKEY$ keystroke input routine, with B$ receiving the value, if any, of INKEY$. The ASCII value of B$ is tested in lines 211–214. Depending on which arrow key is struck, execution jumps to the needed move routine. Line 215 is reached only if the key that was struck was not an arrow key. Lines 230–490 make up four separate move routines. These routines move the required row (variable I) in the required direction, depending on which arrow key was struck. Lines 500 and 510 ask if you wish to play again, and are reached only if you hit the Z key.

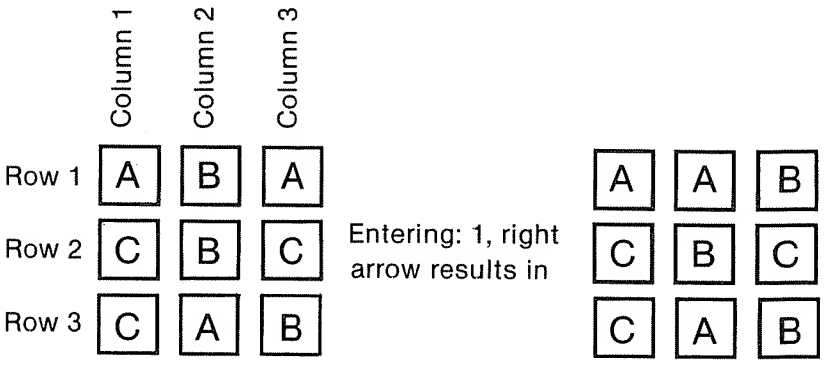


Figure 1. *Explanation of moves*

Lines	Purpose
1–26	rules
28	prints colored blocks
30	prints letter blocks
40–150	shuffle
160–210	input moves
230–280	moves row right
300–350	moves column down
360–420	moves row left
430–490	moves column up

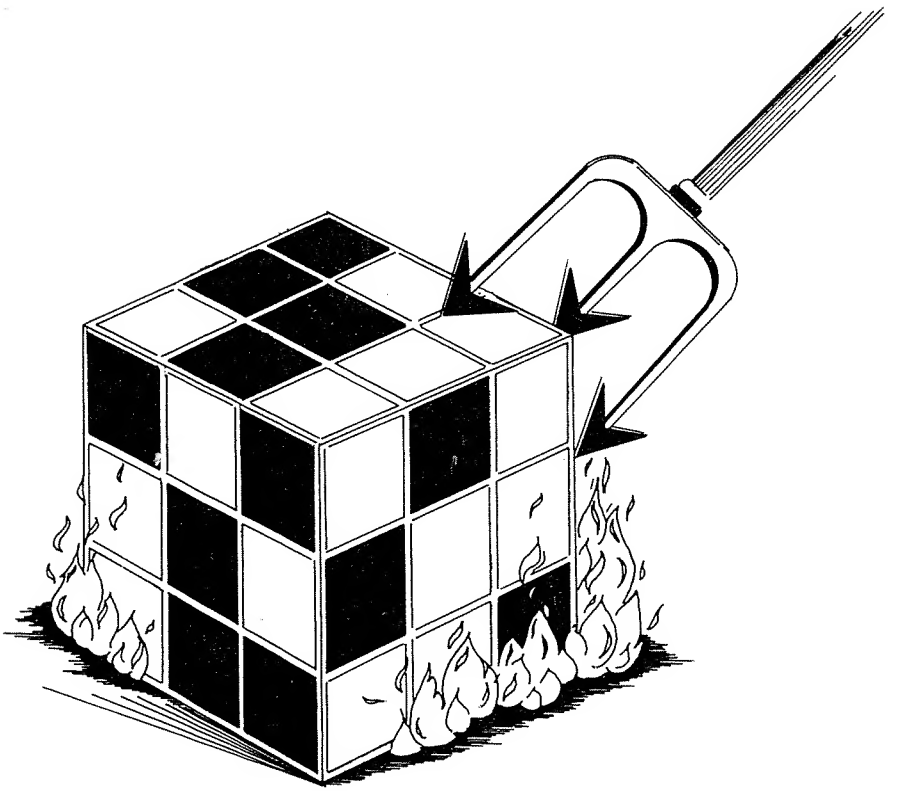
Table 1. *Line descriptions*

Program Listing. *Satan's Square*

```
1 REM JAMES WOOD, 424 N.MISSOURI, ATWOOD, ILL, 61913, SEPT 1981
3 CLS:PRINT:PRINTTAB(8)"WOOD'S SQUARE"
4 PRINT:PRINT" I WILL GIVE YOU A SQUARE":PRINT"PATTERN WITH ROWS
 OF IDENTICAL":PRINT"ITEMS. I CAN SHUFFLE THE":PRINT"PATTERN OR
YOU MAY SHUFFLE.":PRINT"THE OBJECT OF THE GAME IS TO":PRINT"RETU
RN THE PATTERN TO ROWS OF":PRINT"IDENTICAL OBJECTS."
5 PRINT" TO MOVE USE 1,2,3, ETC FOR THE":PRINT"ROW OR COLUMN YOU
 WANT MOVED":PRINT"AND THE ARROWS FOR THE":PRINT"DIRECTION OF MO
VEMENT."
10 PRINT:PRINT"DIFFICULTY 3 TO 8";
15 A$=INKEY$:IFA$=""THEN15ELSEA=VAL(A$)
20 IFA<3 ORA>8 THEN10
25 PRINT:PRINT"COLOR BLOCKS OR LETTER BLOCKS":PRINT"
(C/L)";
26 D$=INKEY$:IFD$="C"THEN28ELSEIFD$="L"THEN30ELSE26
28 CLS0:FORB=1TOA:FORC=1TOA:PRINT@C*2+(B-1)*64,CHR$(143+(B-1)*16
);:NEXTC,B:GOTO40
30 CLS0:FORB=1TOA:FORC=1TOA:PRINT@C*2+(B-1)*64,CHR$(64+B);:NEXTC
,B
40 PRINT@20,"SHUFFLE(Y/N)";
41 A$=INKEY$:IFA$="Y"THEN50ELSEIFA$="N"THEN160ELSE41
50 PRINT@20,STRING$(12,128);
51 REM VERTICAL SHUFFLE
55 FORE=1TOA:FORF=1TORND(A)
60 H=PEEK(960+64*A+2*E)
65 FORD=A-1TO1STEP-1
70 G=PEEK(960+2*E+64*D)
75 POKE1024+2*E+64*D,G:NEXTD
80 POKE1024+2*E,H
85 NEXTF,E
99 REM HORIZONTAL SHUFFLE
100 FORD=1TOA:FORF=1TORND(A)
105 H=PEEK(A*2+64*D+960)
108 FORE=A-1TO1STEP-1
110 G=PEEK(960+E*2+64*D)
120 POKE962+E*2+64*D,G:NEXTE
140 POKE962+64*D,H
150 NEXTF,D
160 PRINT@20,STRING$(12,128);
165 PRINT@411,"MOVES";
169 REM MOVES
170 PRINT@505," ";
171 PRINT@406,X;:X=X+1
190 B$=INKEY$:IFB$=""THEN190
195 IFB$="Z"THEN500
200 I=VAL(B$):IF I<1 OR I>A THEN190
201 PRINT@504,I;
210 C$=INKEY$:IFC$=""THEN210
211 IFASC(C$)=9THEN230
212 IFASC(C$)=8THEN360
213 IFASC(C$)=94THEN430
214 IFASC(C$)=10THEN300
215 GOTO210
230 H=PEEK(A*2+64*I+960)
240 FORE=A-1TO1STEP-1
250 G=PEEK(960+E*2+64*I)
260 POKE962+E*2+64*I,G:NEXTE
270 POKE962+64*I,H
280 GOTO170
300 H=PEEK(64*A+2*I+960)
310 FORD=A-1TO1STEP-1
320 G=PEEK(960+2*I+64*D)
330 POKE1024+2*I+64*D,G:NEXTD
340 POKE1024+2*I,H
350 GOTO170
360 H=PEEK(962+64*I)
370 FORE=1TOA-1
```

Program continued

```
380 G=PEEK(962+E*2+64*I)
390 POKE960+E*2+64*I,G:NEXTE
400 POKE960+A*2+64*I,H
420 GOTO170
430 H=PEEK(1024+2*I)
440 FORD=1TOA-1
450 G=PEEK(1024+2*I+64*D)
460 POKE960+2*I+64*D,G:NEXTD
470 POKE960+64*A+I*2,H
490 GOTO170
500 CLS:PRINT"PLAY AGAIN (Y/N)"
510 A$=INKEY$:IFA$="Y"THENRUNELSEIFA$="N"THENENDELSE510
```



Card Playing

by Louis Zeppa

My interest in programming card games led me to read *Scarne on Cards.* This book by the acclaimed card mastermind discusses rules, strategies, and tactics for various games.

Two of the aspects Scarne emphasizes for games such as rummy, stud poker, blackjack, and pinochle are memory and awareness. According to Scarne, a player should try to remember cards as they are played during each hand. Scarne claims that, through all the shuffles, cuts, and play, he is aware of where cards sit in the deck.

Program Listing 1 begins with a well-managed deck by initializing a full deck (lines 60–80) and randomly disordering it (lines 90–150). You can obtain a presorted deck by removing the remark in line 61.

For shuffling, the deck is split into two equal packs (lines 160–210) and mixed so that the top card moves down to the second position and the bottom card moves up one spot. You can choose any number of shuffles.

Cutting the cards depends on which player is the dealer. The computer does it randomly in its turn (lines 220–280). The player enters the number of cards to be removed from the top of the deck and placed at the bottom. The smaller pile must have at least five cards. Line 225 provides a 20-card cut that may be used with the presorted deck for debugging. Remembering the relative position of specific cards lets you cut the deck to your advantage.

After the two piles are rearranged into a single deck, the hands are dealt. During and after play, the cards are gathered according to the way they are played. After initialization, the only RANDOM function used is for the cut after shuffling.

To demonstrate deck management, I've used two elementary card games. In the first one (see Program Listing 1), known to children as war or jacks (depending on whether aces or jacks are high), all the cards are dealt into piles face down in front of each player (lines 300–330). The GOSUB 5000 in line 330 displays the hands after the deal. Play commences when the players turn over their top cards (line 370). High card takes the trick (lines 430–520), and the winner adds the cards to the bottom of his or her stack (lines 460 and 510). Break ties by drawing two cards (lines 570–580) from each player's stock and matching a third card as described for regular play. The winner is the player who captures all the cards.

The game has no skill element at all, so the program plays both sides without intervention. If this program were for children, not demonstration, there would be graphics cards and player entries. Using MID$ of TRS-80 Disk BASIC would reduce the number of string exchanges and eliminate the

string garbage collection routine but would be incompatible as a demonstration for non-disk computers.

To keep array pointers simple, as top cards are played to tricks, they are put into temporary array MX$( ) in lines 370 and 570–580 and removed from each player's stack. The stack arrays are rearranged so the next card is element 1 of the array (lines 680–690). The game is written as an infinite loop. When one side wins all the cards, the deck is shuffled and cut, and a new game is dealt.

The second game (see Program Listing 2) adds a small tactical element. Each player is dealt four cards (lines 820–860); the rest of the deck is reserved as a stock. Changing variable NP in line 800 allows you to experiment with other sized hands. The designated player (variable C%) leads with a card (lines 1220–1260); the other player must respond with a higher card to win the trick (lines 1270–1330). The winner of the previous trick fills his hand by drawing the top card from the stock (line 1400 or 1420) and leads with a card from his hand. The second player draws the next card and plays to the lead card. The goal of the game is to take one more than half of the tricks—14 tricks for this 52-card deck.

Because this is a demonstration of deck management, the card tactics and strategy used are simple. It takes a higher ranked card to take the trick, so the lead plays the highest ranked card in hand (lines 1200–1260). The response is either the lowest ranked card that will take the trick or the lowest card in the hand (lines 1270–1330).

Cards are taken up in the order played. Unplayed cards are gathered in this order: remainder in winner's hand, remainder in loser's hand, any left in the stock (lines 1470–1490 called from lines 960 and 970). Again, the computer plays both hands. When there is a winner, the computer shuffles, cuts, and deals for another game.

Shuffling Along

I used to wonder how many shuffles would return a deck to the original order. Thinking it would take forever, I've never done it with real cards but decided to try it on the computer. Program Listing 3 creates a new deck (lines 40–50) and shuffles it (lines 80–90) until the original order returns. Counting the number of shuffles (line 100), it saves the top card after each shuffle in array D(H) and counts how many times any card returns to its original position regardless of order (lines 120–140).

In the perfect-mix-shuffle, 52 shuffles restore the deck to its original order. Needless to say, I was surprised. When setting up the experiment, H was set as a double-precision variable, and there was no array of first cards. No card returns to its original position until they all do, but each card goes to each of the other 52 positions along the way. The array of top cards after each shuffle is itself a shuffled deck.

A magazine article I read had another theory of card shuffling. According to "The Mathematics of Card Tricks" (*SCIENCE82*, August 1982, pp. 7 +), eight perfect shuffles will return a deck to its original order. I tested my idea and theirs and found both to be correct. Eight shuffles will restore the order of the deck if the top card remains on top; that is, 51 cards are shuffled instead of 52. This method provides the basis for many card tricks. To modify my program to perform the eight-card shuffle, switch the loop indices of lines 80 and 90 (or 81 and 91) as follows:

```
80 W = 0: FOR X = 1 TO M - 1 STEP 2: W = W + 1: B(X) = A(W): NEXT
90 FOR X = 2 TO M STEP 2: W = W + 1: B(X) = A(W): NEXT
```

I derived from observation a method for finding the position of a card after a shuffle. If M is the number of cards in the deck and X is the position of the card, then two times X will be its position after one shuffle. If two times X is greater than the number of cards in the deck, then its position will equal two times X less the size of the deck plus one or (2*X) – (M + 1). Assuming a 52 card deck, a card's position will move from 1 to 2 to 4 to 8 to 16 to 32. Using the above equation, its next position will move then to 11, (2*32) – (52 + 1), then to 22 to 44 to 35, (44*2) – (52 + 1), and so on.

The preceding example has two important implications for card-playing programs. First, the following programmed equation will enable the computer, or a person, to estimate the location of a card:

```
10 X = (initial location)
20 X = 2*X
30 IF X > (size of deck) THEN X = X - (size of deck + 1)
40 IF (another shuffle) THEN GOTO 20
```

If a program, as player, contains some such routine, as the game progresses it will develop a complete picture of the deck, independent of the computer as deck manipulator.

Second, an array of pointers may be used to reduce or even eliminate actually shuffling a deck's string array. Alternate lines 71, 81, 91, and so on show the use of pointers to manipulate the deck; that is, instead of manipulating strings, the program will manipulate integer arrays which are faster and take up less space. The simplest implementation of that idea would be to specify the number of shuffles:

```
10 INPUT "HOW MANY TIMES SHOULD I SHUFFLE THE DECK"; NS
```

followed by this routine (M equals the size of the deck):

```
20 FOR Z = 1 TO M: X = Z
30 FOR Y = 1 TO NS
40 X = X*2
50 IF X > M THEN X = X - (M + 1)
60 NEXT Y
70 Z(Z) = X
80 NEXT Z
```

Whenever a shuffle is needed, the array pointer Z(Z) is used:

```
90 FOR Z = 1 TO M: A(Z) = B(Z(Z)): NEXT Z
```

Because having the computer remember the deck complicates things, I haven't programmed any complicated card games. I am now developing a winning strategy, and I need a compact method of storing the deck, but such refinements do not belong in a discussion of shuffling.

Program Listing 1

```
10 CLS:CLEAR5000:DEFINTA-Z:DEFSTRA-G:Y$="CDHS":A="12"
20 Z$="23456789TJQKA":M=LEN(Y$)*LEN(Z$):DIMT(LEN(Y$),LEN(Z$))
30 DIMB(M),A(M),I(M),C(M),D(M),MX$(M),J(M)
40 NS=4: ' NUMBER OF SHUFFLES BETWEEN HANDS
50 DEFFNXY(X$,Y$)=INSTR(X$,RIGHT$(Y$,1))
60 PRINT"      CREATE THE DECK"
61 ' GOSUB5130:GOTO290: ' TO USE PRESORTED DECK
70 W=0: FORY=1TOLEN(Y$): FORZ=1TOLEN(Z$): W=W+1
80   B(W)=MID$(Y$,Y,1)+MID$(Z$,Z,1): A(W)=B(W): NEXTZ,Y
90 PRINT"      DISORDER THE DECK"
100 FORX=1TOM: I(X)=0: NEXT
110 FORX=1TOM
120 I=RND(M)
130 IFI(I)=0THENJ(X)=I: I(I)=I ELSE120
140 NEXTX
150 FORX=1TOM: A(X)=B(J(X)): NEXT
160 PRINT"      SHUFFLING ";
161 'GOSUB5150
170 FORZ=1TONS: W=0: PRINTZ;
180  FORX=2TOMSTEP2: W=W+1: B(X)=A(W): NEXT
190  FORX=1TOM-1STEP2: W=W+1: B(X)=A(W): NEXT
200  FORX=1TOM: A(X)=B(X): NEXT
210 NEXTZ: PRINT
220 PRINT"      CUTTING ";
225 'CU%=20:GOTO240: ' PRESET CUT POSITION
230 CU%=RND(M-10)+5
240 PRINT"CUT IS BETWEEN"CU%" AND"CU%+1
250 FORX=1TOM: Y=X+CU%
260  IFY>MTHENCU%=CU%-M: Y=1
270  B(X)=A(Y): NEXT
280 A="":FORX=1TOM: A(X)=B(X): A=A+A(X): NEXT
290 '  CONTINUE POINT FOR THE TWO DIFFERENT GAMES
300 PRINT"     DEALING JACKS"
310 Y=0: FORX=1TOMSTEP2: Y=Y+1
320  C(Y)=A(X): D(Y)=A(X+1)
330 NEXTX: GOSUB5000: ' TO DISPLAY HANDS
340 ' PLAYING
350 R=1: I1=Y: J1=Y: I2=1: J2=1
360 PRINT"C HAS"I1" CARDS; D HAS"J1" CARDS ";
370 MX$(R)=C(1): MX$(R+1)=D(1)
380 I1=I1-1:J1=J1-1:GOSUB680:GOSUB690
390 LC=FNXY(Z$,MX$(R)): LD=FNXY(Z$,MX$(R+1))
400 ' LC=INSTR(Z$,RIGHT$(MX$(R),1))
410 ' LD=INSTR$(Z$,RIGHT$(MX$(R+1),1))
420 PRINT"
C : "MX$(R)" >><< "MX$(R+1)"  : D"
430 '  FIRST EVALUATION
440 IFLC=>LDTHEN490ELSEIFI1<1THEN640
450 '          D HAS HIGHER CARD
460 I=0: FORX=J1+1TOJ1+(R+1):I=I+1:D(X)=MX$(I):NEXT
470 J1=J1+I: R=1: GOTO360
480 '  SECOND EVALUATION
490 IFLC=LDTHEN540ELSEIFJ1<1THEN650
500 '          C HAS HIGHEST CARD
510 I=0:FORX=I1+1TOI1+(R+1):I=I+1:C(X)=MX$(I):NEXT
520 I1=I1+I: R=1: GOTO360
530 '  THIRD EVALUATION
540 IFI1<1ANDJ1<1THEN660
550 '  FOURTH EVALUATION
560 IFI1<1THEN640ELSEIFJ1<1THEN650
570 R=R+2:MX$(R)=C(1):MX$(R+1)=C(2)
580 R=R+2:MX$(R)=D(1):MX$(R+1)=D(2)
590 I1=I1-2
600 IFI1<1THEN640ELSEI2=2:GOSUB680
610 J1=J1-2
620 IFJ1<1THEN640ELSEJ2=2:GOSUB690
630 R=R+2: I2=1: J2=1: GOTO360
```

Program continued

```
640 PRINT"'D' WINS":GOTO670
650 PRINT"'C' WINS":GOTO670
660 PRINT" TIE"
670 GOTO160
680 FORX=1TOI1:C(X)=C(X+I2):NEXT:RETURN
690 FORX=1TOJ1:D(X)=D(X+J2):NEXT:RETURN
4990 END
5000 PRINT"   DISPLAYING HANDS"
5010 FORX=1TOY:PRINTC(X)" ";:NEXT:PRINT
5020 FORX=1TOY:PRINTD(X)" ";:NEXT:PRINT
5030 RETURN
5040 '        VERIFYING NO DUPLICATE CARDS IN DECK
5050 FORX=1TOM:FORY=XTOM:IFX=YTHEN5070
5060 IFA(X)=A(Y)THENPRINTX" "A(X);Y" "A(Y):STOP
5070 PRINTCHR$(145);CHR$(8);:NEXTY,X: RETURN
5080 '        PRESORTED DECK
5090 DATA D8,SK,CJ,H2,C5,C8,D9,CK,S3,SJ,C3,H8,S6
5100 DATA H6,HJ,ST,H9,HT,S9,D4,CT,D2,DT,C9,D3,H5
5110 DATA S7,HK,DQ,CQ,D7,C6,S4,S5,D6,DK,SQ,S8,HA
5120 DATA DA,DJ,H4,C7,H3,C4,S2,D5,H7,CA,HQ,SA,C2
5130 POKE16553,255: FORX=1TOM: READA(X): NEXT:RETURN
5140 ' VERIFY NO DUPLICATE CARDS
5150 A="": FORX=1TOM:A=A+A(X):NEXT:B=A
5160 PRINT:PRINTA
5170 'AY="":AY=INKEY$:IFAY=""THEN10152
5180 RR=0:FORX=1TO2*MSTEP2: KO=0: D=MID$(A,X,2)
5190 IFRIGHT$(D,1)=" "THEN5230ELSEKA=1
5200 KA=INSTR(KA,A,D): IFKA=0THEN5230
5210 MID$(A,KA,2)="A "
5220 KO=KO+1:IFKO>1THENPRINTX;KA" "D:PRINTA:PRINTB:RR=1ELSE5200
5230 NEXTX:IFRR<>1THENRETURN
5240 PRINT"10215";
5250 GOTO5250
```

Program Listing 2

```
10 CLS:CLEAR5000:DEFINTA-Z:DEFSTRA-G:Y$="CDHS":A="12"
20 Z$="23456789TJQKA":M=LEN(Y$)*LEN(Z$):DIMT(LEN(Y$),LEN(Z$))
30 DIMB(M),A(M),I(M),C(M),D(M),MX$(M),J(M)
40 NS=4: ' NUMBER OF SHUFFLES BETWEEN HANDS
50 DEFFNXY(X$,Y$)=INSTR(X$,RIGHT$(Y$,1))
60 PRINT"      CREATE THE DECK"
61  GOSUB5130:GOTO290: ' TO USE PRESORTED DECK
70 W=0: FORY=1TOLEN(Y$): FORZ=1TOLEN(Z$): W=W+1
80   B(W)=MID$(Y$,Y,1)+MID$(Z$,Z,1): A(W)=B(W): NEXTZ,Y
90 PRINT"      DISORDER THE DECK"
100 FORX=1TOM: I(X)=0: NEXT
110 FORX=1TOM
120 I=RND(M)
130 IFI(I)=0THENJ(X)=I: I(I)=I ELSE120
140 NEXTX
150 FORX=1TOM: A(X)=B(J(X)): NEXT
160 PRINT"      SHUFFLING ";
161 GOSUB5150
170 FORZ=1TONS: W=0: PRINTZ;
180  FORX=2TOMSTEP2: W=W+1: B(X)=A(W): NEXT
190  FORX=1TOM-1STEP2: W=W+1: B(X)=A(W): NEXT
200  FORX=1TOM: A(X)=B(X): NEXT
210 NEXTZ: PRINT
220 PRINT"      CUTTING ";
225 'CU%=20:GOTO240: ' PRESET CUT POSITION
230 CU%=RND(M-10)+5
240 PRINT"CUT IS BETWEEN"CU%" AND"CU%+1
250 FORX=1TOM: Y=X+CU%
260  IFY>MTHENCU%=CU%-M: Y=1
270  B(X)=A(Y): NEXT
```

```
280 A="":FORX=1TOM: A(X)=B(X): A=A+A(X): NEXT
290 '  CONTINUE POINT FOR THE TWO DIFFERENT GAMES
800 SF=1:NP=4:NN=1:Y=0:N=2*NP:IFZZ=0THENZZ=1:C%=RND(2)
810 FORX=1TOLEN(Y$):FORR=1TOLEN(Z$):T(X,R)=0:NEXTR,X
820 PRINT"     DEAL"
830 FORX=1TONSTEP2: Y=Y+1
840  IFC%=1THENC(Y)=A(X):D(Y)=A(X+1)
850  IFC%=2THEND(Y)=A(X):C(Y)=A(X+1)
860 NEXT
870 PRINT"     PLAY"
880 NN=N:SC=0:SD=0
890 FORX=1TONP:PRINTC(X)" ";:NEXT:PRINT
900 FORX=1TONP:PRINTD(X)" ";:NEXT:PRINT
910 IFC%=1THENGOSUB1010ELSEGOSUB1120
920 PRINT" SCORE  C:"SC" D:"SD
930 IF(SC<14)AND(SD<14)AND((SD+SC)<M/2)THEN890
940 PRINT"C'S SCORE :"SC,"D'S SCORE :"SD
950 IFSC>SDTHENC%=1ELSEIFSD>SCTHENC%=2ELSEC%=RND(2)
960 IFNP>0THENIFC%=1GOSUB1470:GOSUB1480:ELSEGOSUB1480:GOSUB1470
970 IFNN<MTHENGOSUB1490
980 FORX=1TOM:A(X)=B(X):NEXT:GOTO160
990 END
1000 ' C LEADS
1010 GOSUB1440: GOSUB1230
1020 GOSUB 1450: GOSUB1280
1030 PRINT"C PLAYS "C(YM);
1040 IFYP>=YGORYS=0THENPRINT" D PLAYS "D(YT);ELSEPRINT" D PLAYS "D(YS)
     ;
1050 IFYP>=YGORYS=0THENGOSUB1070:C%=1:RETURN
1060 GOSUB1090:C%=2:RETURN
1070 W$=C(YM):L$=D(YT):GOSUB1350
1080 X=YM:GOSUB1370:X=YT:GOSUB1380:GOSUB1390:RETURN
1090 W$=D(YS):L$=C(YM):GOSUB1350
1100 X=YS:GOSUB1380:X=YM:GOSUB1370:GOSUB1410:RETURN
1110 ' D LEADS
1120 GOSUB 1450: GOSUB1230
1130 GOSUB 1440: GOSUB1280
1140 PRINT"D PLAYS "D(YM);
1150 IFYP>=YGORYS=0THENPRINT" C PLAYS "C(YT);ELSEPRINT" C PLAYS "C(YS)
     ;
1160 IFYP>=YGORYS=0THENGOSUB1180:C%=2:RETURN
1170 GOSUB1200:C%=1:RETURN
1180 W$=D(YM):L$=C(YT):GOSUB1350
1190 X=YM:GOSUB1380:X=YT:GOSUB1370:GOSUB1410:RETURN
1200 W$=C(YS):L$=D(YM):GOSUB1350
1210 X=YS:GOSUB1370:X=YMN:GOSUB1380:GOSUB1390:RETURN
1220 ' FIRST PLAY
1230 YP=0:YR=14:FORX=1TONP
1240 Y=FNXY(Z$,MX$(X))
1250 IFY>YPTHENYP=Y:YM=X
1260 NEXT: RETURN
1270 ' RESPONSE
1280 YS=0: YG=14: YH=14
1290 FORX=1TONP
1300  Y=FNXY(Z$,MX$(X))
1310   IFY>YPANDY<YGTHENYG=Y:YS=X
1320   IFY<=YPANDY<YHTHENYH=Y:YT=X
1330 NEXT: RETURN
1340 ' ADD USED CARDS TO TAKE-UP STOCK
1350 B(SF)=W$:B(SF+1)=L$:SF=SF+2:RETURN
1360 ' REARRANGE HANDS AFTER PLAY
1370 FORX=XTONP-1:C(X)=C(X+1):NEXT:RETURN
1380 FORX=XTONP-1:D(X)=D(X+1):NEXT:RETURN
1390 IFNN>=MTHENNP=NP-1:SC=SC+1:RETURN
1400 C(NP)=A(NN+1):D(NP)=A(NN+2):NN=NN+2:SC=SC+1:RETURN
1410 IFNN>=MTHENNP=NP-1:SD=SD+1:RETURN
1420 D(NP)=A(NN+1):C(NP)=A(NN+2):NN=NN+2:SD=SD+1:RETURN
1430 ' PUT HANDS IN TEMPORARY ARRAY
1440 FORX=1TONP:MX$(X)=C(X):NEXT:RETURN
```

Program continued

```
1450 FORX=1TONP:MX$(X)=D(X):NEXT:RETURN
1460 '  FILL DECK WITH UNPLAYED CARDS
1470 FORX=1TONP:B(SF)=C(X):SF=SF+1:NEXT:RETURN
1480 FORX=1TONP:B(SF)=D(X):SF=SF+1:NEXT:RETURN
1490 FORX=SFTOM:B(X)=A(NN+1):NN=NN+1:NEXT:RETURN
5000 PRINT"   DISPLAYING HANDS"
5010 FORX=1TOY:PRINTC(X)" ";:NEXT:PRINT
5020 FORX=1TOY:PRINTD(X)" ";:NEXT:PRINT
5030 RETURN
5040 '        VERIFYING NO DUPLICATE CARDS IN DECK
5050 FORX=1TOM:FORY=XTOM:IFX=YTHEN5070
5060 IFA(X)=A(Y)THENPRINTX" "A(X);Y" "A(Y):STOP
5070 PRINTCHR$(145);CHR$(8);:NEXTY,X: RETURN
5080 '        PRESORTED DECK
5090 DATA D8,SK,CJ,H2,C5,C8,D9,CK,S3,SJ,C3,H8,S6
5100 DATA H6,HJ,ST,H9,HT,S9,D4,CT,D2,DT,C9,D3,H5
5110 DATA S7,HK,DQ,CQ,D7,C6,S4,S5,D6,DK,SQ,S8,HA
5120 DATA DA,DJ,H4,C7,H3,C4,S2,D5,H7,CA,HQ,SA,C2
5130 POKE16553,255: FORX=1TOM: READA(X): NEXT:RETURN
5140 ' VERIFY NO DUPLICATE CARDS
5150 A="": FORX=1TOM:A=A+A(X):NEXT:B=A
5160 PRINT:PRINTA
5170 'AY="":AY=INKEY$:IFAY=""THEN10152
5180 RR=0:FORX=1TO2*MSTEP2: KO=0: D=MID$(A,X,2)
5190 IFRIGHT$(D,1)=" "THEN5230ELSEKA=1
5200 KA=INSTR(KA,A,D): IFKA=0THEN5230
5210 MID$(A,KA,2)="A "
5220 KO=KO+1:IFKO>1THENPRINTX;KA" "D:PRINTA:PRINTB:RR=1ELSE5200
5230 NEXTX:IFRR<>1THENRETURN
5240 PRINT"10215";
5250 GOTO5250
```

Program Listing 3

```
10 CLS:CLEAR5000:DEFINTA-Z:DEFSTRA-G:Y$="CDHS":A="12"
20 Z$="23456789TJQKA":M=LEN(Y$)*LEN(Z$):DIMZ(M),K(M),X(M),Y(M)
30 DIMA(M),B(M),C(M),D(M):FORZ=1TOM:A(Z)=A:B(Z)=A:C(Z)=A:NEXT
40 W=0: FORY=1TOLEN(Y$): FORZ=1TOLEN(Z$): W=W+1
50  B(W)=MID$(Y$,Y,1)+MID$(Z$,Z,1): A(W)=B(W): NEXTZ,Y
60 FORZ=1TOM: C(Z)=B(Z): X(Z)=Z: Y(Z)=Z: NEXT
70 FORZ=1TOM: A(Z)=B(Z): NEXT
71 'FORZ=1TOM: Z(Z)=Y(Z): NEXT
80 W=0:FORX=2TOMSTEP2:W=W+1:B(X)=A(W):NEXT
81 'W=0:FORX=2TOMSTEP2:W=W+1:Y(X)=Z(W):NEXT
90 FORX=1TOM-1STEP2:W=W+1:B(X)=A(W):NEXT:GOSUB170
91 'FORX=1TOM-1STEP2:W=W+1:Y(X)=Z(W):NEXT:GOSUB171
100 H=H+1:D(H)=B(1):FORZ=1TOM: PRINTB(Z)"   "C(Z),
101 'H=H+1:K(H)=Y(1):FORZ=1TOM:PRINTA(Y(Z))"  "A(Z(Z)),
110 IFB(Z)=C(Z)THENNEXTZ:GOTO150ELSEPRINTH;L
111 'IFA(Y(Z))=A(X(Z))THENNEXTZ:GOTO150ELSEPRINTH;L
120 FORZ=1TOM
130 IFB(Z)=C(Z)THENL=L+1:PRINTL;H" "B(Z)"/"C(Z)
131 'IFA(Y(Z))=A(X(Z))THENL=L+1:PRINTL;H" "A(Y(Z))"/"A(X(Z))
140 NEXT:GOTO70
150 PRINT"IT TOOK"H" SHUFFLES AND THERE WERE"L" RETURNS."
160 FORZ=1TOM:PRINTZ" "D(Z),:NEXT: END
161 'FORZ=1TOM:PRINTZ" "A(K(Z)),:NEXT: END
170 FORZ=1TOM:PRINTZ;B(Z)"-"C(Z),:NEXT:RETURN
171 'FORZ=1TOM:PRINTZ;A(Z(Z))"-"A(X(Z)),:NEXT:RETURN
```

Another Magic Trick

by David D. Busch

This magic trick uses a subtle communications technique that is unlikely to be spotted by anyone but another amateur magician. In other words, you may safely repeat this trick without fear of exposure by spoilsports in your crowd.

The trick works as follows: The Wizard explains that mental communication between humans is amplified by the mere presence of a computer in the room. To demonstrate, he runs Computer ESP (see Program Listing). Those in the room can operate the program themselves, while the Wizard either turns his back or leaves the room entirely.

First, the computer asks the participants to enter the names of eight common objects found in the room. These can be entered in any order. Next, the players are offered the choice of choosing one of the objects for the magician to locate, or of having the computer do the selecting. In the latter case, the TRS-80 keeps the choice secret until the Wizard has revealed his own prediction.

At this point, the Wizard returns to the room to see the selection of eight objects on the screen. He should make a point of never touching the computer and comment to the audience that mere proximity to the computer is contact enough to amplify his innate ESP powers. The Wizard then writes down a prediction on a piece of paper, seals it in an envelope, and hands it to an onlooker. Someone who knows what the choice is announces the name of the object, or, if the computer has chosen, someone presses a key to reveal the verdict. When the Wizard's sealed envelope is opened, the prediction is shown to be 100 percent correct.

The audience is sure to be baffled, even if the trick is run several times. There are no obvious ways for the computer to have communicated to the Wizard. The only thing that appears on the screen is the message, *Okay, Wizard, See if you can find the right object!*, and a list of the objects themselves. The wording of the message remains the same each time, precluding any secret codes. The objects are listed in the same manner every time the program is run. Unless a very astute observer comes along, the magician's secret is safe.

The Secret

The secret is in the seemingly unchanging message at the top of the screen. Each time it is printed, there is one space after each word of the

message, except one word, which is followed by two spaces. The extra space is unnoticeable to anyone not looking for it but sticks out like a sore thumb to someone in the know. The position of the space points to which object has been chosen, either by the computer, or by the players; that is, if the space follows the first word, object number one has been chosen.

Most good magic tricks have a very simple premise, and this one is no different. What transforms it into something special is the patter you develop to make the effect seem very mysterious. To enhance the program, delete the listing on the screen that is visible to the magician. Instead, have each object placed under a bowl or hat marked with numbers. By glancing only casually at the message on the screen, the prestidigitator can determine which hat the selected object is under, without even knowing which object has been chosen.

How the Program Works

In lines 10–40, the individual parts of the message are read into a string array, SE$(n), and stored for later use as sentence segments. After the instructions are displayed, a FOR-NEXT loop of 1 to 8 asks for the names of eight objects, which are stored in OB$(n). These objects are listed on the screen by lines 200–230, and the players offered the choice of selecting one or of having the computer do the job. They may input a number (stored in variable NU), or the computer will choose NU through the RND function (line 330).

The screen clears, and the seemingly innocent message appears on the screen, from a subroutine at lines 390–460. The FOR-NEXT loop prints each word of the sentence with only one space following, except if the loop counter (N3) equals NU. In that case, an extra space is added.

The names of the objects are listed to the screen next. If you want to use my suggested enhancement, delete lines 480–500. For sessions in which the computer has chosen the target object (CH = 2), a message, *Only the computer knows for sure!* appears. Nothing happens on the screen until the magician reveals his choice. Then, OB$(NU) is unveiled, and much acclamation and applause result from the appreciative onlookers.

Program Listing. *Computer ESP*

```
5 CLEAR 400
10 DATA "Okay,","Wizard,","See","if","you","can",
        "find","the","right","object!"
20 :  FOR N=1 TO 10
30 :    READ SE$(N)
40 :  NEXT N
50 CLS:PRINT TAB(20) "  C O M P U T E R  E S P "
60 PRINT
70 PRINT "  While the Wizard is not looking, you may"
80 PRINT "  enter the names of eight common objects "
90 PRINT "  contained within this room.  One may then"
100 PRINT "  be selected as the object which the Wizard"
110 PRINT "  will attempt to locate by extra-sensory means."
120 PRINT
130 PRINT "    Hit any key when ready to select objects  :"
140 IF INKEY$="" GOTO 140
150 :  FOR N1=1 TO 8
160 :    CLS:PRINT:PRINT
170 :    PRINT "Enter the name of object #";N1;", contained in this ro
    om."
180 :    INPUT OB$(N1)
190 :  NEXT N1
200 CLS:PRINT
210 :  FOR N2=1 TO 8
220 :    PRINT TAB(10) N2;".) ";OB$(N2)
230 :  NEXT N2
240 PRINT
250 PRINT " Would you like to :"
260 PRINT "                    1.) Select object yourself"
270 PRINT "                    2.) Have computer select object"
280 PRINT "                        (and keep choice to itself)"
290 PRINT
300 CH$=INKEY$:IF CH$="" GOTO 300
310 CH=VAL(CH$)
320 IF CH<1 OR CH>2 GOTO 300
330 IF CH=2 THEN NU=RND(8): GOTO 380
340 PRINT  " Enter number of object to be located :";NU$
350 NU$=INKEY$:IF NU$="" GOTO 350
360 NU=VAL(NU$)
370 IF NU<1 OR NU>8 GOTO 350
380 CLS:PRINT:PRINT
390 :  FOR N3=1 TO 10
400 :    IF N3<>NU GOTO 430
410 :    SE$=SE$(N3)+" "
420 :    GOTO 440
430 :    SE$=SE$(N3)
440 :    PRINT SE$;
450 :    PRINT CHR$(32);
460 :  NEXT N3
470 PRINT:PRINT
480 :  FOR N4=1 TO 8
490 :    PRINT N4;".) ";OB$(N4)
500 :  NEXT N4
510 PRINT
520 IF CH=2 PRINT "Only the computer knows for sure !"
530 PRINT
540 PRINT " After Wizard has revealed choice, hit any key."
550 IF INKEY$="" GOTO 550
560 CLS:PRINT:PRINT:PRINT " The chosen object is :"
570 PRINT:PRINT OB$(NU);" ! "
580 GOTO 580
```

GRAPHICS

Graphics and ZBASIC
Part I
Part II
Part III

Unlocking the Color Computer
Graphics Character Code
SBLOCK

Graphics and ZBASIC Part I

by John Corbani

Sooner or later, anyone who gets hooked on programming has to try a graphics package. I've been the proud owner of a TRS-80 Model I for almost two years now and finally have the graphics in fair shape. Unfortunately, the combination of low resolution and slow speed made patience the first requirement to do anything interesting. Then the ZBASIC compiler was developed. It is a solid, interactive piece of software. Integer arithmetic and strings are supported along with most of the BASIC commands. It improves the speed, and, if you move things fast enough, the resolution isn't so bad after all.

These articles cover a set of graphics primitives, data base handling, dimensional transforms, and real-time operator interaction from the keyboard. The programs are developed first as BASIC routines and then modified to be compatible with ZBASIC. All programs are based on a Level II 16K TRS-80 Model I and 16K ZBASIC. The programs will also run on a 16K Model III.

Two graphics modes are possible with the TRS-80: the character mode and the point mode. The character mode uses the standard alphanumeric screen layout with 1024 locations. This mode can be used to create high-speed movement of quite complex characters and is best used to handle predefined images.

The point mode breaks each character position on the screen into six individually addressable points. The total number of points then becomes 6144. This is enough to demonstrate a wide variety of arbitrarily defined graphics figures.

The display screen is arranged as 128 horizontal (x-axis) by 48 vertical (y-axis) points. The x-axis points are numbered from 0 on the left to 127 on the right. The y-axis points are numbered from 0 at the top to 47 at the bottom. The aspect ratio of each point is roughly 2 to 1. That is, each point is twice as high as it is wide.

The BASIC commands that apply to these points are SET(X,Y), RESET(X,Y), and POINT(X,Y). SET turns the selected point white. RESET turns the selected point black. POINT is a variable with a value of 0 if the selected point is black, and a value of -1 if the point is white. The x- and y-coordinates must be limited to displayable points, or the interpreter returns an error message and halts operation.

Three primitive functions are required of any graphics package:
1) Clear the screen.
2) Move to a point.
3) Draw a line to a second point.

The first is a BASIC function; the second can be implemented by setting variables; the third is best handled as a subroutine. Define all variables as integers before use. I chose variables that are easily remembered mnemonics that are compatible with ZBASIC.

Clearing the Screen

The BASIC command CLS turns the screen black. It is possible to clear the screen to white and plot using black on white. The finished picture looks fine, but black streaks cover the screen during plotting. This series of articles will lead to interactive graphics that allows continuous plotting. White on black is the best choice.

Moving to a Point

The Draw subroutine uses X1 and Y1 as the starting point for all plotting. The calling program must set X1 and Y1 before calling the Draw subroutine if the new line must start anywhere other than the end of the previous line.

Drawing a Line

The Draw subroutine uses X2 and Y2 as the ending point for all lines. The Draw function includes a clipping routine to check for lines outside the screen boundaries. Once a line is determined to be plottable, each point is checked so that only legal points are SET. All lines are drawn with a maximum error of 1/2 point with no rounding off error at the end points. Once a line is complete, X1,Y1 is made equal to X2,Y2, and control returns to the calling program.

Let's review the ground rules for drawing a line between two points on a grid. Lines and corners should be smooth. Lines should start and end on the exact x- and y-coordinates called for. Lines should be symmetrical, within one point, end for end. Lines should be continuous. Coordinates must be easy to use. Transforming coordinates from system to system and clipping are generally eased by defining the center of the screen as X0, Y0. X values to the left of center are minus, to the right, plus. Y values below center are minus, above center, plus. Plotting must be fast. A graphics package is never fast enough.

The Draw subroutine is written at line 950 and starts by checking for lines that are unplottable. The center of the screen is 0,0. Plottable positions are plus or minus 63 points horizontally, by plus or minus 23 points vertically.

```
950 IF X1>63 AND X2>63 THEN 1070
960 IF X1<-63 AND X2<-63 THEN 1070
```

```
970 IF Y1>23 AND Y2>23 THEN 1070
980 IF Y1<-23 AND Y2<-23 THEN 1070
1070 X1=X2: Y1=Y2: RETURN
```

Line 1070 is the last line of the routine. X1 and Y1 are set up to allow the drawing of contiguous vectors by resetting X2,Y2 and calling 950 again. If the program makes it past line 980, at least some part of the line will probably be visible on the screen and must be plotted. To get smooth lines, the program must decide whether the distance between endpoints is greater along the x-axis or the y-axis. It must then single step along the longer axis from the starting to the ending point. At each step it determines the value of the other axis and plots the point. Speed requirements argue against the use of floating point math, multiplication, or division. The following algorithm is not commonly known, but performs the function using only addition and subtraction.

```
1000 D1=X2-X1: D2=Y2-Y1: S1=SGN(D1): S2=SGN(D2): D1=ABS(D1):
     D2=ABS(D2): E=0: E1=D1+D1: E2=D2+D2: IF D1<D2 THEN 1050
1010 FOR D=0 TO D1: GOSUB 1080: X1=X1+S1: E=E+E2:
     IF E>=D1 THEN Y1=Y1+S2: E=E-E1
1020 NEXT D: GOTO 1070
1050 FOR D=0 TO D2: GOSUB 1080: Y1=Y1+S2: E=E+E1:
     IF E>=D2 THEN X1=X1+S1: E=E-E2
1060 NEXT D
```

Line 1000 determines Delta X and Delta Y and which is bigger. S1 and S2 are the sign and size of the steps to be taken along the two axes. E1 and E2 are used to determine when to step the shorter axis. E keeps track of how things are going. Now all there is to do is to check if the point is on the screen, transform the coordinates, and plot the point.

```
1080 IF ABS(X1)<64 AND ABS(Y1)<24 THEN SET(63+X1,23-Y1)
1090 RETURN
```

The subroutine above can be demonstrated by entering and running Program Listing 1, which draws a star on the screen. The demonstration program in BASIC runs in 15.5 seconds, and almost all of the time is used to draw lines. This is very slow. Save the program as G.

On to ZBASIC

The first order of business is to compile the Draw subroutine. The endpoints of the lines can be determined in BASIC and POKEd into memory. The BASIC program can then call the compiled plot routine with USR(0). 16K ZBASIC uses the memory locations shown in Table 1 to store the variables.

ZBASIC allows the AND function only in the arithmetic sense. In lines 950 to 980 and line 1080, you must change the AND to THEN IF.

```
950 IF X1>63 THEN IF X2>63 THEN 1070
```

```
960 IF X1<-63 THEN IF X2<-63 THEN 1070
970 IF Y1>23 THEN IF Y2>23 THEN 1070
980 IF Y1<-23 THEN IF Y2<-23 THEN 1070
1080 IF ABS(X1)<64 THEN IF ABS(Y1)<24 SET(63+X1,23-Y1)
```

Change line 10 to show the name GRAPHICS Z1 Z.

```
10 REM                GRAPHICS Z1 "Z"          7/6/81 JC
```

Delete lines 20 through 80 and save the BASIC program as Z. Now compile and remember that you cannot use line feeds in a program that will be compiled. Use spaces for formatting. If there are no compile errors, save the compiled program as Z1 using the ZBASIC SAVE. If a mistake is made later, reloading is faster than retyping. Return to BASIC.

All ZBASIC compiled programs start at 25165. This converts to a low-byte value of 37 and a high-byte value of 98. BASIC stores the USR(0) calling address at 16526 and 16527. The VARPTR function is used to aid in passing variables. VARPTR(A1) gives the address of the low byte of the BASIC variable (A1). The high byte follows. Variables A1 and A2 are used to transfer data from place to place. This demonstration program is shown in Program Listing 2.

Save the program as G. Now run it. It takes less than one second per star.

Variable	Low Byte	High Byte
X1	32686	32687
Y1	32688	32689
X2	32750	32751
Y2	32752	32753

Table 1. *Variables and memory addresses*

Program Listing 1

```
10 REM              GRAPHICS 1 "G"           7/6/81 JC
20 DEFINT A,D,E,S,X,Y
50 DATA -30,-17, 0,23, 30,-17, -45,8, 45,8, -30,-17
60 CLS: RESTORE: READ X1,Y1
70 FOR A=1 TO 5: READ X2,Y2: GOSUB 950: NEXT
80 FOR A=1 TO 1000: NEXT: GOTO 60
950 IF X1>63 AND X2>63 THEN 1070
960 IF X1<-63 AND X2<-63 THEN 1070
970 IF Y1>23 AND Y2>23 THEN 1070
980 IF Y1<-23 AND Y2<-23 THEN 1070
1000 D1=X2-X1: D2=Y2-Y1: S1=SGN(D1): S2=SGN(D2): D1=ABS(D1):
     D2=ABS(D2): E=0: E1=D1+D1: E2=D2+D2: IF D1<D2 THEN 1050
1010 FOR D=0 TO D1: GOSUB 1080: X1=X1+S1: E=E+E2:
     IF E>=D1 THEN Y1=Y1+S2: E=E-E1
1020 NEXT D: GOTO 1070
1050 FOR D=0 TO D2: GOSUB 1080: Y1=Y1+S2: E=E+E1:
     IF E>=D2 THEN X1=X1+S1: E=E-E2
1060 NEXT D
1070 X1=X2: Y1=Y2: RETURN
1080 IF ABS(X1)<64 AND ABS(Y1)<24 THEN SET(63+X1,23-Y1)
1090 RETURN
```

Program Listing 2

```
10 REM              GRAPHICS 2  "G"        7/6/81
20 DEFINT A,X,Y: A1=0: A2=0: AA=VARPTR(A1):
   AB=AA+1: AC=VARPTR(A2): AD=AC+1: POKE 16526,37:
   POKE 16527,98
30 XA=32686: XB=XA+1: YA=32688: YB=YA+1:
   XC=32750: XD=XC+1: YC=32752: YD=YC+1
50 DATA  -30,-17,  0,23,  30,-17, -45,8, 45,8, -30,-17
60 CLS: RESTORE: READ A1,A2: GOSUB 210
70 FOR A=1 TO 5: READ A1,A2: GOSUB 220: NEXT
80 FOR A=1 TO 1000: NEXT: GOTO 60
210 POKE XA,PEEK(AA): POKE XB,PEEK(AB):
    POKE YA,PEEK(AC): POKE YB,PEEK(AD):
    RETURN
220 POKE XC,PEEK(AA): POKE XD,PEEK(AB):
    POKE YC,PEEK(AC): POKE YD,PEEK(AD):
    X=USR(0): RETURN
```

Graphics and ZBASIC Part II

by John Corbani

The first part of this series brought a TRS-80 to the point at which lines, defined by their X,Y endpoints, could be plotted at a reasonable speed. This part will cover objects, an object world, and a viewer looking at that world.

Objects to be manipulated by a graphics package may be described by a series of lines defining the object's outlines and features. The endpoints of these lines form a graphics data base which describes objects existing in a world coordinate system.

The ability to manipulate data is limited by the integer math of ZBASIC. No number can exceed an absolute value of 32767. The requirements of multiplying two numbers reduce the practical limits of a graphics world to absolute values of 250 or less. The world may be defined as a cube 500 units per side. The cube has a front, back, top, bottom, left side, and right side. The x-axis goes from −250 on the left to +250 on the right. The y-axis goes from −250 on the front to +250 on the back. The z-axis goes from −250 on the bottom to +250 on the top. X(0), Y(0), Z(0) is the center of the world.

A viewer can look at the world from anywhere inside the world. The viewer's position can be defined as an X,Y,Z point. The viewing direction must be defined in pitch, yaw, and roll. Pitch is the viewing angle above or below the X,Y plane. Straight up is +90 degrees. Level is 0 degrees. Straight down is −90 degrees. Yaw is the angle to the left or right of the Z,Y plane. The yaw angle is 0 degrees, when looking from front to back of the world. The angle can be either minus (left) or plus (right). The roll axis is not varied in this series of programs. The viewer's head is assumed to have no tilt to the left or right.

This program (see Program Listing) sets up a data base describing a small house. The house comes complete with a picture window, front door, and flagpole in the side yard. The viewer's position and viewing angles are obtained from the operator. The house and flagpole are then transformed into two-dimensional space and displayed on the TRS-80 screen.

The objects to be worked with are described by lines, and a line requires a beginning and an end. Even lines of zero length each require two X,Y,Z coordinates. Lines with bends in them require an additional X,Y,Z coordinate for each bending point. The data base must contain a flag that indicates the end of a line. The program must also know when the entire data

base has been processed. Since the largest value allowed in the data base is 250, 254 is used to indicate the end of a line, and 255 indicates the end of the data base.

A demonstration program may be written now to illustrate the use of a simple data base holding three-dimensional data. The program uses the Graphic 2 G (Part 1) subroutines at lines 210 and 220 to pass variables to the ZBASIC line drawing routines and call them. If ZBASIC is not available, add (D,E,S) to the DEFINT statement in line 20. Change (GOSUB 210:) in line 70 to (X1 = A1: Y1 = A2). Change (GOSUB 220:) in line 80 to (X2 = A1: Y2 = A2: GOSUB 950:). Delete the (XY = USR(0)) in line 220. Use the plot subroutine starting at line 950 in the Graphics 1 G (Part 1) program. Everything will work, but it takes a while.

```
10 REM              GRAPHICS 5  "G"         7/13/81
20 DEFINT A,B,C,J,K,X,Y,Z: DIM D(150): A1=0: A2=0: A3=0:
   JA=16: KA=0: X=-20: Y=-100: Z=10:
   AA=VARPTR(A1): AB=AA+1: AC=VARPTR(A2): AD=AC+1:
   POKE 16526,37: POKE 16527,98
30 XA=32686: XB=XA+1: YA=32688: YB=YA+1:
   XC=32750: XD=XC+1: YC=32752: YD=YC+1
40 DATA
  -12,-8,0, -12,8,0, -12,8,12, -12,0,18, -12,-8,12, -12,-8,0,
   12,-8,0,  12,8,0,  12,8,12,  12,0,18,  12,-8,12,  12,-8,0,
   254,
  -12,-8,12, 12,-8,12, 254,
  -12,0,18, 12,0,18, 254,
  -12,8,12, 12,8,12, 254,
  -12,8,0, 12,8,0, 254
42 DATA
  -7,-8,3, -7,-8,7, 0,-8,7, 0,-8,3, -7,-8,3, 254,
   4,-8,0, 4,-8,7, 7,-8,7, 7,-8,0, 254
45 DATA
   25,-10,0, 25,-10,25, 26,-14,24, 25,-10,23, 254, 255
50 READ D(A): IF D(A)<>255 THEN A=A+1: GOTO 50 ELSE CLS: A=0
```

All integer variables have been defined and the compiled routine's calling address has been entered. The array D(n) has been filled with the object data. The viewer's data must now be obtained.

```
60 PRINT @ 0, "DIR" JA TAB(12) "PITCH" KA TAB(24) "X POS" X
   TAB(36) "Y POS" Y TAB(48) "Z POS" Z;:
   PRINT @ 66,;: INPUT JA: PRINT @ 80,;: INPUT KA:
   PRINT @ 92,;: INPUT X: PRINT @ 104,;: INPUT Y:
   PRINT @ 116,;: INPUT Z
65 L=JA*.01745: J1=SIN(L)*50+.5: J2=COS(L)*50+.5:
   L=-KA*.01745: K1=SIN(L)*50+.5: K2=COS(L)*50+.5:
   CLS: A=0
```

No checking of operator input is performed in this program. Numbers above 250 may cause unpredictable results, but no permanent damage. JA

and KA are converted from degrees to radians immediately. The sine and the cosine of the angles are required to handle rotation of the image. This forces the use of floating point arithmetic and slows things down, but there is little alternative when using BASIC. Multiplying by 50 turns the sines and cosines to integers with two percent accuracy. This is good enough for most objects to be transformed properly.

The next lines of code scan through the data base, have the data transformed and plotted, and then look for operator input.

```
70 A1=D(A): B1=D(A+1): C1=D(A+2): GOSUB 110: GOSUB 210: A=A+3
80 A1=D(A): B1=D(A+1): C1=D(A+2):
   IF B1<>255 THEN
   IF A1=254 THEN A=A+1: GOTO 70 ELSE
   GOSUB 110: GOSUB 220: A=A+3: GOTO 80
90 GOTO 60
```

The Transform Routine calculates the difference in distance between the viewing and object points in all planes. The x- and y-coordinate values are individually corrected for rotation. The values are then divided by the distance to obtain the correct image size. Note that a correction is made to adjust perspective to a reasonable level. A correction is also made for the 2 to 1 aspect ratio of the TRS-80 graphics points. This allows the screen image to remain the same size no matter what the viewer's orientation.

```
110 U=A1-X: U1=B1-Y: U2=C1*-1+Z:
    T=U*J2-U1*J1: T1=U1*J2*K1/50+U*J1*K1/50-U2*K2:
    T2=U1*J2*K2/50+U*J1*K2/50+U2*K1:
    A1=T/T2*300: A2=T1/T2*150: RETURN
```

A1 and A2 are now plottable x- and y-coordinates. They must be passed to the plotting routine as either X1,Y1 or X2,Y2. Save the program as G. Now run it. Enter data on request or hit ENTER to pass over coordinates that do not need changing. Plotting the whole data base takes about 10.5 seconds. Too slow! There are a few things that will speed up the BASIC part of the program. Pulling out all the stops (and spaces) can get the time down to eight seconds. It's not enough, and the code is unreadable.

Program Listing

```
10 REM              GRAPHICS 5  "G"        7/13/81
20 DEFINT A,B,C,J,K,X,Y,Z: DIM D(150): A1=0: A2=0: A3=0:
   JA=16: KA=0: X=-20: Y=-100: Z=10:
   AA=VARPTR(A1): AB=AA+1: AC=VARPTR(A2): AD=AC+1:
   POKE 16526,37: POKE 16527,98
30 XA=32686: XB=XA+1: YA=32688: YB=YA+1:
   XC=32750: XD=XC+1: YC=32752: YD=YC+1
40 DATA
  -12,-8,0, -12,8,0, -12,8,12, -12,0,18, -12,-8,12, -12,-8,0,
   12,-8,0,  12,8,0,  12,8,12,  12,0,18,  12,-8,12,  12,-8,0,
   254,
  -12,-8,12, 12,-8,12, 254,
  -12,0,18, 12,0,18, 254,
  -12,8,12, 12,8,12, 254,
  -12,8,0, 12,8,0, 254
42 DATA
  -7,-8,3, -7,-8,7, 0,-8,7, 0,-8,3, -7,-8,3, 254,
   4,-8,0, 4,-8,7, 7,-8,7, 7,-8,0, 254
45 DATA
   25,-10,0, 25,-10,25, 26,-14,24, 25,-10,23, 254, 255
50 READ D(A): IF D(A)<>255 THEN A=A+1: GOTO 50 ELSE CLS: A=0
60 PRINT @ 0, "DIR" JA TAB(12) "PITCH" KA TAB(24) "X POS" X
   TAB(36) "Y POS" Y TAB(48) "Z POS" Z;:
   PRINT @ 66,;: INPUT JA: PRINT @ 80,;: INPUT KA:
   PRINT @ 92,;: INPUT X: PRINT @ 104,;: INPUT Y:
   PRINT @ 116,;: INPUT Z
65 L=JA*.01745: J1=SIN(L)*50+.5: J2=COS(L)*50+.5:
   L=-KA*.01745: K1=SIN(L)*50+.5: K2=COS(L)*50+.5:
   CLS: A=0
70 A1=D(A): B1=D(A+1): C1=D(A+2): GOSUB 110: GOSUB 210: A=A+3
80 A1=D(A): B1=D(A+1): C1=D(A+2):
   IF B1<>255 THEN
   IF A1=254 THEN A=A+1: GOTO 70 ELSE
   GOSUB 110: GOSUB 220: A=A+3: GOTO 80
90 GOTO 60
110 U=A1-X: U1=B1-Y: U2=C1*-1+Z:
    T=U*J2-U1*J1: T1=U1*J2*K1/50+U*J1*K1/50-U2*K2:
    T2=U1*J2*K2/50+U*J1*K2/50+U2*K1:
    A1=T/T2*300: A2=T1/T2*150: RETURN
210 POKE XA,PEEK(AA): POKE XB,PEEK(AB):
    POKE YA,PEEK(AC): POKE YB,PEEK(AD):
    RETURN
220 POKE XC,PEEK(AA): POKE XD,PEEK(AB):
    POKE YC,PEEK(AC): POKE YD,PEEK(AD):
    XY=USR(0): RETURN
```

Graphics and ZBASIC Part III

by John Corbani

The second part of this series got a TRS-80 to the point where you could pretend you were a helicopter pilot. The helicopter could be positioned and oriented to look at any part of a graphics world. Operator input was accomplished by typing numeric values in response to system prompts for the X, Y, and Z positions, direction, and pitch. Plotting the picture took over 10 seconds. This is too slow to easily introduce direct operator interaction, such as joysticks or customized key functions. This final section shows how to compile the transform routines and get the plot time to under two seconds. A keyboard driver is also added to allow interaction.

The BASIC program spends a large amount of time processing the data base as well as transforming each point. The POKE command is especially slow. The first thing you must do is get the data base available to a compiled routine. ZBASIC does not support arrays directly, but it is easy to make your own. ZBASIC has 800 bytes set aside for strings. This is more than enough to handle the data base. The data can be in a BASIC program, POKEd into memory once, and then used as required. The array consists of two-byte integers starting at 31750. About 300 bytes are required by the demonstration program.

The viewing position, as well as the sine and cosine of the viewing angles, is determined in BASIC. Only 14 bytes of data must be passed to plot a new picture. This is reasonable, and besides, ZBASIC has no trigonometric functions. I tried using a look-up table for sine and cosine, but this method was no faster than passing data from BASIC.

The transform routine is severely hampered by the limitations of integer arithmetic. The use of brackets to define what happens when, and adding four extra divisions, minimizes overflows. The multiplier of 50 on the angles is used to advantage. The test in line 115 prevents division by zero and removes all lines that extend behind the viewer. Add the following lines to the Graphics Z1 Z program from Part 1.

```
10 REM                GRAPHICS Z3 "Z"          7/24/81 JC
20 B=31750
70 FOR A=0 TO 5: POKE 32640+A,PEEK(B+A): NEXT A:
   IF B1=255 THEN 90 ELSE IF A1=254 THEN B=B+2: GOTO 70 ELSE
   B=B+6: GOSUB 110: IF P=0 THEN 70 ELSE X1=A1: Y1=A2
80 FOR A=0 TO 5: POKE 32640+A,PEEK(B+A): NEXT A:IF B1<>255 THEN
   IF A1=254 THEN B=B+2: GOTO 70 ELSE B=B+6: GOSUB 110:
```

```
   IF P=0 THEN 70 ELSE X2=A1: Y2=A2: GOSUB 950: GOTO 80
90 RETURN
110 U=A1-X: U1=B1-Y: U2=C1*-1+Z:
    T=U*J2-U1*J1: T1=U1*((J2*K1)/50)+U*((J1*K1)/50)-U2*K2:
    T2=((U1*J2)/50*K2)/50+((U*J1)/50*K2)/50+(U2*K1)/50
115 T2=T2/2: IF T2<1 THEN P=0: RETURN ELSE
    P=1: A1=(T*2)/T2: A2=T1/T2: RETURN
```

The original lines from 950 to 1090 plot the line after the endpoints have been calculated. Remember that the line formatting is done with spaces. No line feeds are allowed in ZBASIC. CSAVE the program as Z. Compile the program. If no errors are reported, save the compiled program as Z3 using the ZBASIC SAVE command. Return to BASIC.

You must modify the Graphics 5 G program from Part 2 to load the data base and pass the viewer's data to the compiled display program. Lines 40 to 65 should be saved. Delete everything else and add the following lines:

```
10 REM              GRAPHICS 6  "G"          7/11/81
20 DEFINT A,J,K,X,Y,Z: DIM D(150): J1=0: J2=0: K1=0: K2=0:
   JA=16: KA=0: X=-20: Y=-100: Z=10:
   AA=VARPTR(J1): AB=VARPTR(J2):
   AC=VARPTR(K1): AD=VARPTR(K2):
   AE=VARPTR(X):  AF=VARPTR(Y): AG=VARPTR(Z)
30 JM=32658: JN=32722: KM=32660: KN=32724:
   XA=32622: YA=32624: ZA=32626: B=31750:
   POKE 16526,37: POKE 16527,98
50 READ J1: POKE B, PEEK(AA): POKE B+1, PEEK(AA+1):
   IF J1<>255 THEN B=B+2 : GOTO 50 ELSE CLS
230 POKE JM,PEEK(AA): POKE JM+1,PEEK(AA+1):
    POKE JN,PEEK(AB): POKE JN+1,PEEK(AB+1):
    POKE KM,PEEK(AC): POKE KM+1,PEEK(AC+1):
    POKE KN,PEEK(AD): POKE KN+1,PEEK(AD+1)
240 POKE XA,PEEK(AE): POKE XA+1,PEEK(AE+1):
    POKE YA,PEEK(AF): POKE YA+1,PEEK(AF+1):
    POKE ZA,PEEK(AG): POKE ZA+1,PEEK(AG+1)
250 XY=USR(0): GOTO 60
```

CSAVE the program as G and run it. If there have been no errors, the house should plot in about 1.7 seconds. There have been a lot of changes, and if an error occurs, first try to find out if it is in the BASIC program or the ZBASIC program. Edit BASIC as required. If the problem appears to be in the compiled routines, run Graphics 6 G to load all the memory locations with known data. Load Graphics Z3 Z and check for errors there. Edit, save, and recompile using SYSTEM (ENTER) /22528 (ENTER) to call the compiler. Once this is complete, save the compiled program and return to BASIC. Test the program by using XY = USR(0) in the immediate mode. When all is well, reload Graphics 6 G.

A response time of less than two seconds is fast enough to allow interactive manipulation of the graphics data base. Eight of the TRS-80 keys can be

modified to simulate the action of a helicopter's controls. Two others can be used to tilt the viewing angle up and down. The control stick is simulated with E for forward, X for back, S for left, and D for right. Collective pitch is simulated with T for up and B for down. The rudder (tail rotor) pedals are simulated by K for left and L for right. Pressing I tilts the viewing angle up, and a comma (,) tilts the viewing angle down.

The subroutine which handles the keyboard input uses the INKEY$ function to check for key activity. Once a key is detected, five of the keyboard memory bytes are stored. Tests are made in turn for each key. The viewer variables are updated, a picture is plotted, and the keyboard is scanned again. Any number of keys can be held down simultaneously, and the imaginary helicopter will respond accordingly. Acceleration is infinite since all motion stops when all keys are released.

Load Graphics 6 G, delete line 65, and enter the following new lines:

```
10 REM              GRAPHICS 7  "G"         7/12/81
260 ZI=0: KI=0
270 JI=0: XI=0: YI=0
280 I$=INKEY$: IF I$="" THEN 280
290 I1=PEEK(14337): I2=PEEK(14338): I3=PEEK(14340):
    I4=PEEK(14344): I5=PEEK(14368)
300 IF (I1 AND 16)=16 THEN XI=XI+1
310 IF (I3 AND 8)=8 THEN XI=XI-1
320 IF (I1 AND 32)=32 THEN YI=YI+1
330 IF (I4 AND 1)=1 THEN YI=YI-1
340 IF (I3 AND 16)=16 THEN ZI=ZI+10
350 IF (I1 AND 4)=4 THEN ZI=ZI-10
360 IF (I2 AND 2)=2 THEN KI=KI+10
370 IF (I5 AND 16)=16 THEN KI=KI-10
380 IF (I2 AND 16)=16 THEN JI=JI+10
390 IF (I2 AND 8)=8 THEN JI=JI-10
400 IF XI<>0 THEN X=X+XI*J2/5: Y=Y-XI*J1/5
410 IF YI<>0 THEN Y=Y+YI*J2/5: X=X+YI*J1/5
420 JA=JA+JI: Z=Z+ZI
430 IF ABS(KA+KI)<91 THEN KA=KA+KI
450 GOTO 65
```

CSAVE the program and then run it. The keyboard routine only added a tenth of a second to the time, so updates are still less than two seconds each. Practice with the program until you are comfortable with the controls and then delete line 280. The program will now update the picture continuously. Remove line 270. The helicopter is now in free flight. There is momentum forward and back, left and right, and in rotation. Try flying around the house about 150 feet out. Establish some velocity to the right and start rotation to the left. A nice orbit can be established. When you get good with these axes, delete line 260.

This program is a good base for a lot of simulations. Add instrumentation

if you wish. Altimeter reads Z, rate of climb reads ZI, airspeed reads XI, and so on. Just print the numbers wherever you wish on the screen. You could change the helicopter to a space ship and create worlds to explore.

Everything is not taken care of in such a short program. The math can overrun, and there is no true clipping of the data. Lines extending behind your back are not plotted, even though a portion of them should be visible. Even so, the program always returns for new input. The next step is yours.

Unlocking the Color Computer Graphics Character Code

by David R. Barr

In a recent article in *80 Microcomputing* ("Unlocking the Graphic Code," June 1981, page 147), Jerome Weintraub showed how a simple binary code was the basis of the ASCII character codes for the graphics characters on the TRS-80 Model I computer. I was inspired to do the same for the TRS-80 Color Computer.

First, I looked at the graphics character codes on the reference card that comes with the Color Computer. There I found the pattern shown in Figure 1.

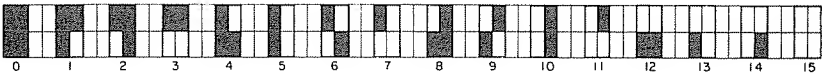


Figure 1. *Graphics character codes*

Following Mr. Weintraub's suggestion, I tried adding pattern 1 and pattern 2 to obtain pattern 3 and noted that the non-black parts of the patterns for 1 and 2 combined to form the non-black part of the pattern for 3; a check of the patterns for 4, 9, and 13 showed that the conjecture held (see Figure 2). I then checked each pattern that had only one non-black sector, and noted that these were patterns 1, 2, 4, and 8—the powers of 2. Now the Color Computer graphics character code was broken; using Figure 3, add the sector numbers corresponding to the non-black sectors of the graphics character.

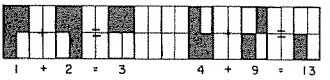


Figure 2. *Adding patterns*

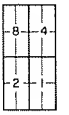


Figure 3. *Breaking the code—add the sector numbers corresponding to the non-black sectors of the graphics character.*

Now that I had broken the code for the pattern number, the rest was easy. The reference card even had a formula:

$$\text{code} = 128 + 16*(\text{color} - 1) + \text{pattern}$$

I thought that it would be awkward to have to subtract one from the color

number every time, and since this formula could be written as:

code = 128 + 16*color – 16 + pattern

I decided to use the equivalent formula:

code = 112 + 16*color + pattern.

Next, I sought an application for my discovery. A friend let me play with his Space Invaders game by Spectral Associates of Tacoma, Washington, and I noticed that the words "Space Invaders" appear in large, colored letters that look like dot matrix printing using graphics characters. This inspired me to use my discovery to print dot matrix symbols on the screen using the graphics characters.

I decided to use a 5×7 dot matrix of sectors. Since the graphics characters are all 2×2 sectors, this meant that I had to use a 6×8 matrix of sectors, which is a 3×4 matrix of graphics characters. The extra row and column of sectors serve as a space between symbols and between rows.

Figure 4 shows the dot matrix for the letter A. Adding up the values for the sector numbers for the 12 graphics characters resulted in the following calculations:

```
2 + 4 + 8 = 14        4 = 4        1 + 2 + 4 + 8 = 15
    1 + 8 = 9     1 + 2 + 8 = 11       1 + 4 + 8 = 13
        1 = 1         1 + 2 = 3            1 + 4 = 5
1 + 2 + 4 = 7 1 + 2 + 4 + 8 = 15       1 + 2 + 4 = 7
```

To print out the letter A, you print patterns 14, 4, and 15 on one line, patterns 9, 11, and 13 below them, patterns 1, 3, and 5 on the next line down, and patterns 7, 15, and 7 on the last line.

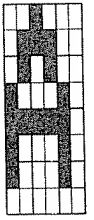


Figure 4. *Dot matrix for the letter A*

I have written a short program (see Program Listing) which illustrates some of the possibilities for this dot matrix printing using graphics characters. The program prints out a copyright statement in a fancy form. Line 20 is a DATA statement, with numbers 14, 4, 15, 9, 11, 13, 1, 3, 5, 7, 15, and 7 as data. You should recognize these as the numbers of the patterns for the letter A. Lines 30 through 60 contain the pattern numbers for some other symbols. In lines 80 through 140, these five symbols are read into an array A which is dimensioned (5,3,4). The first dimension of array A denotes the symbol, the second dimension of array A denotes the row, and the third

dimension of array A denotes the column. When you write your own program, the first number in the dimension statement of array A would be the number of symbols you are using, while the other two numbers would be unchanged.

Line 70 is a DATA statement specifying which symbols are to be printed (symbol 4 first, then symbol 3, symbol 5 next, etc.). This data is read into an array B, dimensioned (8), in lines 150 through 170. You can fit only 10 symbols per line using this size dot matrix. When you write your own program, you might find it convenient to give this array a second dimension to keep track of which line is being printed.

The program cycles through the eight colors in lines 180 and 450, and there are two printings per color—one in black characters against a colored background, and the second in colored characters against a black background. Line 190 clears the screen, leaving a colored background, and lines 240 through 300 print black letters on this colored background. Line 330 clears the screen to black, and lines 340 through 420 print the colored letters on the black background. Only one line is printed in black against a colored background, while a maximum of three lines can be printed in color against a black background. The index Q is used to cycle through these three lines of print.

Lines 270 and 380 contain the PRINT@ instructions which are the heart of each algorithm. There are two parts to consider: the calculation of the ASCII code of the graphics character to be printed, and the calculation of the location at which to print it.

Let us consider first the easier of the two calculations, that of the ASCII code of the graphics character to be printed. We know that it can be calculated using the formula 112 + 16*color + pattern. Since C is the color number, it follows that all but the pattern number is given by 112 + C*16, and that the pattern number is given by A(B(T),R,S) in line 270 and by 15 – A(B(T),R,S) in line 380, where T is the symbol order number, R is the row number, and S is the column number. Since array B contains the numbers of the symbols in the order that they are to be printed, it follows that B(1) is the first symbol number (here symbol 4), B(2) is the second symbol number (here symbol 3), and so on. Thus, A(B(T),R,S) is the pattern number stored in row R and column S of the array for the symbol B(T), and use of the pattern number given by A(B(T),R,S) will give a black letter on a colored background. Since subtracting a pattern number from 15 gives the number of the pattern with black and non-black sectors reversed, line 380 contains 15 – A(B(T),R,S) to give a colored symbol on a black background.

The order of the three FOR-NEXT loops using T, R, and S can be set up in six different ways. Two are illustrated in the program. For black symbols on a colored background, the order R-T-S from outer loop to inner loop prints out the top quarter of all the symbols in the line first, then prints out the sec-

ond quarter of all of the symbols, followed by the third and fourth quarters; no symbol is complete until the printing of the last cycle on R. For colored symbols on a black background, the order T-S-R from outer loop to inner loop prints out each symbol completely before starting the next symbol. When you write your own program, you will have to choose which order you wish to use.

Next, we consider the more complicated of the calculations in the PRINT@ statements—the calculation of the location at which to print the graphics character. Since each row is of length 32, adding 32 to a previous location places a character directly below the previous location in the next row. For this reason, we multiply the row number, R, by 32, to obtain the R*32 part of the calculation. Characters in the same row are in successive locations; therefore we use the column number, S, to assign these successive locations. Symbols on the same line are three locations apart, so we multiply the symbol order number, T, by 3.

We now have R*32 + S + T*3 as the basis for calculating the location. When R, S, and T are all equal to 1, this places the first character in location 32 + 1 + 3 = 36, and the first symbol printed will occupy locations 36, 37, 38, 68, 69, 70, 100, 101, 102, 132, 133, and 134 if nothing else is done to shift the locations. In line 270, this is where the first symbol printed is located. In line 380, another starting location is used, and three lines are printed rather than just one. The index of the lines is Q, which goes from 0 to 2 for lines 1 to 3. Since each line uses four rows on the screen, each of length 32, adding 128 to a previous location places a character in the same relative position in the next line. For this reason, we multiply the line number by 128 to obtain Q*128.

All that remains to be explained is the calculation of the starting location for each line. The program was written so as to have the three lines staggered. Each is eight symbols long in a line of length 10. First the two blanks are on the right, then there is one at each end, and finally they are both on the left. Since Q starts at 0, the starting values of Q, R, S, and T will still place the first character in location 36. To place the first character in location 32, we must subtract 3, which is done by subtracting 1 from T before multiplying by 3. The final feature of the calculation in line 380 staggers the starting position in each line. This is accomplished by adding Q to T – 1 prior to multiplication by 3. When you write your own program, you can use the basic R*32 + S + T*3 augmented by the calculation to place the start of the symbol where you want it.

The restriction to using no more than three lines of at most 10 symbols does limit the applications to things like printing out "Klingons Destroy Enterprise" in emphatic letters at the end of an unsuccessful game of Star Trek. These large size symbols can be intermingled with ordinary size ASCII symbols by using PRINT@ or other print commands, as is done at the start

of the program. This cannot be done with symbols generated in one of the graphics modes described in "Do-it-yourself #7-5" of Radio Shack's *Going Ahead With Extended Color Basic.*

The unlocking of the color computer graphics character code opens up a new frontier, and I hope that this sample program will inspire you to use this potential in a creative way.

Program Listing

```
10 DIM A(5,4,3),B(8)
20 DATA 14,4,15,9,11,13,1,3,5,7,15,7
30 DATA 1,3,13,4,12,7,5,15,5,3,3,15
40 DATA 15,15,15,15,15,15,12,15,15,3,15,15
50 DATA 1,6,15,5,15,5,5,14,7,3,7,15
60 DATA 1,3,13,4,12,7,1,13,15,7,11,15
70 DATA 4,3,5,3,2,1,5,5
80 FOR X=1 TO 5
90 FOR Y=1 TO 4
100 FOR Z=1 TO 3
110 READ A(X,Y,Z)
120 NEXT Z
130 NEXT Y
140 NEXT X
150 FOR BN=1 TO 8
160 READ B(BN)
170 NEXT BN
180 FOR C=1 TO 8
190 CLS(C)
200 IF C<>1 GOTO 240
210 PRINT@7,"COPYRIGHT 1981 BY"
220 FOR U=1 TO 100
230 NEXT U
240 FOR R=1 TO 4
250 FOR T=1 TO 8
260 FOR S=1 TO 3
270 PRINT@R*32+S+T*3,CHR$(A(B(T),R,S)+112+C*16);
280 NEXT S
290 NEXT T
300 NEXT R
310 FOR T1=1 TO 500
320 NEXT T1
330 CLS(0)
340 FOR Q=0 TO 2
350 FOR T=1 TO 8
360 FOR S=1 TO 3
370 FOR R=1 TO 4
380 PRINT@R*32+S+(T-1+Q)*3+Q*128,CHR$(15-A(B(T),R,S)+112+C*16);
390 NEXT R
400 NEXT S
410 NEXT T
420 NEXT Q
430 FOR T1=1 TO 500
440 NEXT T1
450 NEXT C
460 GOTO 180
```

SBLOCK

by Jeff Collins

Programs using good graphics are fun to use, but looking up the codes for each graphics block usually puts us off from writing them ourselves. In addition, the large number of decimal graphics values needed in data statements strains limited memory space.

Before running either of the following BASIC programs, assemble and load the program SBLOCK (Program Listing 1) into protected memory by answering the memory size prompt, then using the SYSTEM command. When prompted for an execution address, type /ENTER.

This executes the initialization part of the code, causing the USR ( ) vector to be loaded with the start address of the program as computed during assembly.

The first two programs give you the following capabilities:

1) To create graphics with a bare minimum of hassle.

2) To print screen images faster than POKEing loops and more simply than the packed string.

3) To store more graphics images within the same amount of program memory.

4) To erase alpha/graphics without clearing the entire screen, without complex control code formulas, and without a computed string of spaces.

5) To print sophisticated images without assembly-language programming knowledge.

The program uses only BASIC statements to create screen displays, requiring only that the short assembly program SBLOCK be resident in protected memory. Use the PRINT @ statement to draw any alpha/graphics images, rather than POKE addresses or time-consuming SET/RESET operations. All other BASIC functions are unchanged and available for use at the same time. SBLOCK can be used while debounce or other utilities are in use.

Block-Oriented Strings

The usual way a string is printed is from left to right, from the first byte of the string to the last, printed in contiguous screen memory locations. Using SBLOCK, you can use strings for graphics which are block-oriented, defining the number of columns to print the string bytes across, then continuing on the next line for the same number of columns, and so on, until the last byte. It's like choosing the screen line length for each string you print, rather than always using the built-in screen length of 64 characters per line.

The assembly program knows how many columns to use for the string because the first byte of the string contains this value. The BASIC graphics builder program generates this and the rest of the string's bytes automatically (see Program Listing 2). It lets you change graphics as you go along. But it also keeps you updated on the current column and row location (x and y positions on the graphics layout sheet) at all times, enabling you to enter precise images from the layout sheet.

The other BASIC program (Program Listing 3) is a demonstration of the screen-printing characteristics of the SBLOCK program, which must be resident in protected memory when either of the BASIC programs are run. It is designed not to dazzle you, but rather to show the characteristics of SBLOCK and give you ideas for its use in your own programs. The program presents 13 graphics images as you call for them from the keyboard. By moving the cursor around and flashing the images on and off, you get an idea of how you can use overlapping images to create interesting screen effects.

When you create a graphics block using the builder program, you are asked whether you want the all-blank code spaces to be destructive or nondestructive of those character positions. This, in effect, is asking if you want the spaces within your graphic to erase what is already on the screen in those positions. The answer you give to the prompt depends on your plans for presenting other screen information. If the implications of this seem a little hazy, try playing with the demonstration program. An idea for using such nondestructive spaces in a program should come rather quickly. Try using all the various key commands with each image.

Operation of the Builder Program

Both BASIC programs contain instructions, but it should be noted that the program which builds graphics has two basic modes, graphics and alphanumeric. The graphics mode is the pivotal mode from which all other commands are executed, such as processing an image on the screen into a string, saving the string to tape, or reading one in. The program starts off in the graphics mode and returns there after each function is completed (or alpha mode line is entered). While in the graphics mode, if you press the up-arrow key and choose the graphics mode from the resulting prompt words, the program will leave whatever row and column it was in and return to pointing to row and column zero.

The program begins in the upper left-hand corner, the point from which all your graphics should be created, in order to avoid adding unnecessary space codes to the graphics block string.

Should you choose the alpha mode after hitting the up arrow, simply press any printable character key, including the arrow keys. When done entering alpha information on that line, push the ENTER key. Note that you cannot back space to erase a mistake while in the alpha mode. If you need to erase,

return to the graphics mode by hitting the ENTER key, then back space twice to erase the error. Now that the character is erased, while in the graphics mode, tap the up-arrow key to get the mode prompt, choose the alpha mode, and resume alpha input where you left off. It seems a little strange at first, but it becomes almost automatic after you do it a few times, being similar to getting into and out of an edit mode subcommand in BASIC. This information and that on the screen, combined with a little practice, will keep you from having any trouble at all. Just don't press the BREAK key while there is a graphic on the screen.

How to Use SBLOCK in Your Own Programs

SBLOCK has two printing modes, draw and erase, which are called from a BASIC program. To draw a string block, get into that mode by passing the value 1 from the BASIC program to the assembly routine, as in X = USR(1). To print the string, pass the VARPTR address of the string to be printed, as in X = USR(VARPTR(A$)). To erase the printed string block, pass the erase mode value of zero through the USR argument, again followed by passing the string's VARPTR address. The SBLOCK routine remains in either the draw or the erase mode until the other mode is selected. Take a look at the BASIC program listings to see examples of their use.

A Closer Look at Printing Strings

When you print a regular string of bytes containing control codes, as with printing a CHR$ statement, BASIC interprets each byte of the string. If a control code is encountered, it does a little machine-language routine, such as clearing to the end of the line or screen. If it is a printable character code for that particular machine, it places that value onto the screen.

The program SBLOCK does its own interpreting and printing of a string. It assumes finding printable character codes, except for:

1) The first string byte, always containing the column count.
2) A string-byte value of zero, meaning to skip a video location (nondestructive space) rather than print an actual space there (destructive), whether in the draw or the erase mode.
3) A string-byte value of 128, signaling that the next string byte contains the number of times to repeat printing the following byte value.

An example would be a string of four bytes, with values of 5, 128, 10, and 65. The first byte indicates that the column length should be five. The byte value 128 indicates that byte value 65 should be printed 10 times on the screen, within the column length constraint of five columns. If the fourth byte had a value of zero, then 10 character locations on the video would have been passed over, leaving two screen lines of column length five just as they were before. Fortunately, the graphics string-builder program does all this.

If you wanted, you could change the repeat byte code by changing every statement in the build program which contains CHR$(128) to a different value, one whose graphic you can do without.

The build program does not put code 128 into a string anyway (it becomes space code 32), so I thought it was a good choice to use, leaving all of the Model III non-blank special graphics character codes available for manually inserting within string block DATA statements.

The only disadvantage to using code 128 is that it uses three character spaces within DATA statement lines, whereas the code 1, for example, only uses up one character space each time it is used.

If you decide to change the repeat code in the build program, you must change lines 430, 440, 510, 515, and 810. In the SBLOCK assembly program, change line 420. Having made those changes, if you still want the demonstration program to work, you must change all DATA statement values of 128 to whatever code you decided was expendable. On the Model I, this may be any value from 1 through 31, and the program otherwise works the same as before.

The listings work as is for both the Model I and Model III. Disk BASIC programs have to have a DEFUSR statement added near the beginning; USR0-9 statements must be used in place of the Level II USR statement format, wherever it is used. Also, note in the assembly listing that the initialization code is not needed when Disk BASIC is used to run the BASIC programs.

Note that the cursor is actually left at the same screen position when it is done printing the block graphic image as it was before printing it. At this point in the program, you might want to overlay another graphics string image on the top of the one just printed. You might want to erase it immediately or even save the value of the PRINT@ position in a simple (scalar) variable (or even several consecutive ones in an array) for later exact reference for reprinting or erasing.

I feel that the safest way to plot out any graphics, by whatever method, is by starting with a PRINT@ statement, followed by a null string and a semicolon in the case of printing a block graphic string. That way you can always be certain that your printing will follow your screen layout sheet as closely as possible.

Using SBLOCK is certainly not a cure-all for writing programs with graphics but it does give you the ability to easily print out and erase a specific graphics area. In most cases, it will save you program bytes. Some unnecessary graphics statements will not take up valuable program space. It's also much easier to conceptualize and organize what is to be printed and where it will go on the screen, using unambiguous statements. In any event, it's nice to have another option for displaying appealing graphics.

Program Listing 1. *SBLOCK*

Encyclopedia Loader

```
                00010 ;    *** SBLOCK ***    STRING BLOCK VIDEO PRINT ROUTINE
7530            00020          ORG     30000          ;MAY BE CHANGED TO ANY UNUSED PROTECTED MEM
1A19            00030 READY    EQU     1A19H          ;'READY' PROMPT ADDRESS
7530 213975     00040 INIT     LD      HL,START       ;IF BASIC DISK USE STAT'S
7533 228E40     00050          LD      (USRVEC),HL    ;DEFUSR0-9 AND USR0-9'S
7536 C3191A     00060          JP      READY          ;IN BASIC PROGR'S INSTEAD
7539 CD7F0A     00070 START    CALL    USRARG         ;USR( ) ARG. INTO HL SUBR
753C 7C         00080          LD      A,H            ;MSB OF USR( ) ARGUMENT
753D FE00       00090          CP      0              ;IS IT 0?
753F 2019       00100          JR      NZ,STRADR      ;NO. VARPTR STRING ADDR.
7541 7D         00110          LD      A,L            ;SIGNAL PASSED, NOT ADDR.
7542 FE00       00120          CP      0              ;THE 'ERASE' SIGNAL?
7544 200A       00130          JR      NZ,DRAW        ;NO. MUST BE 'DRAW'
7546 213E20     00140          LD      HL,LDSPAC      ;LABEL VALUE=LD A,' '
7549 227F75     00150          LD      (ERASE),HL     ;DISPLACES NOP'S IN CODE
754C 229C75     00160          LD      (ERAS2),HL     ;DISPLACE MORE NOP'S
754F C9         00170          RET                    ;BACK TO BASIC
7550 210000     00180 DRAW     LD      HL,0           ;EQUALS TWO NOP'S
7553 227F75     00190          LD      (ERASE),HL     ;DRAW NOP'S BACK IN CODE
7556 229C75     00200          LD      (ERAS2),HL     ;DRAW NOP'S HERE TOO
7559 C9         00210          RET                    ;BACK TO BASIC
755A 4E         00220 STRADR   LD      C,(HL)         ;GET ACTUAL STRING LENGTH
755B 0D         00230          DEC     C              ;MINUS COLUMN LENGTH BYTE
755C 23         00240          INC     HL             ;POINT TO LSB OF ADDR.
755D 5E         00250          LD      E,(HL)         ;LSB OF STRING ADDR.
755E 23         00260          INC     HL             ;POINT TO MSB OF ADDR.
755F 56         00270          LD      D,(HL)         ;MSB OF STRING ADDR.
7560 1A         00280          LD      A,(DE)         ;COLUMN BYTE FIRST
7561 32D575     00290          LD      (COLMCT),A     ;SAVE COLUMN COUNT
7564 13         00300          INC     DE             ;PT TO 1ST PRINTABLE BYTE
7565 DD2A2040   00310          LD      IX,(CURSOR)    ;CURRENT CURSOR LOCATION
7569 DDE5       00320          PUSH    IX             ;ONTO STACK,
756B FDE1       00330          POP     IY             ;INTO IY AS SCREEN PTR.
756D D5         00340          PUSH    DE             ;STRING PTR TO STACK,
756E E1         00350          POP     HL             ;INTO HL REG PR.
756F CDC075     00360          CALL    COLUMN         ;GET COLUMN COUNT INTO B
7572 CDC775     00370 INNER    CALL    SCRTST         ;TEST SCREEN LIMITS
7575 D8         00380          RET     C              ;TO BASIC IF CARRY
7576 7E         00390 STRBYT   LD      A,(HL)         ;GET BYTE FROM STRING
7577 FE00       00400          CP      0              ;NON-DESTRUCTIVE SPACE?**
7579 280D       00410          JR      Z,NOLOAD       ;SKIP. NO VIDEO LOAD
757B FE80       00420          CP      128            ;REPEAT CODE ?
757D 2815       00430          JR      Z,REPEAT       ;YES
757F 0000       00440 ERASE    DEFW    0              ;IS LD A,' ' FOR 'ERASE'
7581 CDC775     00450          CALL    SCRTST         ;CHECK SCREEN LIMITS
7584 D8         00460          RET     C              ;TO BASIC IF OUTSIDE
7585 FD7700     00470          LD      (IY+0),A       ;INTO VIDEO DISPLAY
7588 23         00480 NOLOAD   INC     HL             ;PT TO NEXT STRING BYTE
7589 FD23       00490          INC     IY             ;PT TO NEXT VIDEO LOC'N
758B 0D         00500          DEC     C              ;STRING BYTE COUNT
758C C8         00510          RET     Z              ;RETURN TO BASIC IF DONE
758D 10E3       00520          DJNZ    INNER          ;DECREMENT COLUMN COUNT
758F CDB575     00530          CALL    ADD64          ;SCREEN PTR. DOWN A LINE
7592 18DE       00540          JR      INNER          ;GO TEST FOR SCREEN LIMIT
7594 23         00550 REPEAT   INC     HL             ;PT TO REPEAT COUNT BYTE
7595 56         00560          LD      D,(HL)         ;COUNT INTO REG D
7596 23         00570          INC     HL             ;PT TO BYTE TO REPEAT
7597 7E         00580          LD      A,(HL)         ;LOOK AT REPEAT BYTE
7598 FE00       00590 AGAIN    CP      0              ;NON-DESTRUCTIVE CODE?***
759A 2809       00600          JR      Z,ADVANC       ;YES. NO SCREEN OUTPUT
759C 0000       00610 ERAS2    DEFW    0              ;LD A,' ' IF ERASE MODE
759E CDC775     00620          CALL    SCRTST         ;TEST VIDEO LIMITS
75A1 D8         00630          RET     C              ;TO BASIC IF OUTSIDE.
75A2 FD7700     00640          LD      (IY+0),A       ;NO. PRINTABLE CHARACTER
75A5 FD23       00650 ADVANC   INC     IY             ;PT TO NEXT SCREEN LOC'N
75A7 05         00660          DEC     B              ;COLUMN COUNT
75A8 CCB575     00670          CALL    Z,ADD64        ;DROP DOWN A LINE
75AB 15         00680          DEC     D              ;REPEAT COUNT
```

```
75AC 20EA      00690          JR      NZ,AGAIN        ;NOT DONE REPEATING
75AE 0D        00700          DEC     C               ;STRING COUNT REFLECTED
75AF 0D        00710          DEC     C               ;BY REPEAT BYTE, # TO
75B0 0D        00720          DEC     C               ;REPEAT, THE BYTE ITSELF.
75B1 C8        00730          RET     Z               ;TO BASIC, NO MORE BYTES
75B2 23        00740          INC     HL              ;PT TO NEXT STRING BYTE
75B3 18C1      00750          JR      STRBYT          ;GO PROCESS IT
75B5 D5        00760 ADD64    PUSH    DE              ;SAVE
75B6 114000    00770          LD      DE,64           ;ONE LINE OF COLUMNS
75B9 DD19      00780          ADD     IX,DE           ;SCREEN PTR DOWN A LINE
75BB D1        00790          POP     DE              ;RESTORE
75BC DDE5      00800          PUSH    IX              ;NEW SCREEN PTR ADDRESS
75BE FDE1      00810          POP     IY              ;INTO PROPER REG PR.
75C0 E5        00820 COLUMN   PUSH    HL              ;SAVE
75C1 21D575    00830          LD      HL,COLMCT       ;PT TO COLUMN COUNT BYTE
75C4 46        00840          LD      B,(HL)          ;COLUMN COUNT INTO REG B
75C5 E1        00850          POP     HL              ;RESTORE
75C6 C9        00860          RET                     ;BACK TO CALLER
75C7 E5        00870 SCRTST   PUSH    HL              ;SUBR. TESTS VIDEO RANGE
75C8 D5        00880          PUSH    DE              ;SAVE
75C9 FDE5      00890          PUSH    IY              ;VIDEO PTR
75CB D1        00900          POP     DE              ;INTO DE FOR TEST.
75CC 21FF3F    00910          LD      HL,16383        ;HIGHEST VIDEO POSSIBLE
75CF B7        00920          OR      A               ;CLEAR CARRY FLAG
75D0 ED52      00930          SBC     HL,DE           ;HIGHEST - CURRENT PTR
75D2 D1        00940          POP     DE              ;RESTORE
75D3 E1        00950          POP     HL              ;RESTORE
75D4 C9        00960          RET                     ;TO CALLER WITH FLAGS
75D5 00        00970 COLMCT   DEFB    0               ;HOLDS COLUMN COUNT BYTE
4020           00980 CURSOR   EQU     4020H           ;ADDR. HOLDS CURSOR ADDR.
0A7F           00990 USRARG   EQU     0A7FH           ;ADDR OF USR( ) ARG SUBR.
408E           01000 USRVEC   EQU     408EH           ;USR( ) VECTOR ADDR.
203E           01010 LDSPAC   EQU     203EH           ;CODE FOR LD A,' '
7530           01020          END     INIT            ;IF NOT DISK BASIC*******
00000 TOTAL ERRORS
```

Program Listing 2. *BASIC graphics builder*

```
10 REM **** PROGRAM TO BUILD STRING BLOCK  ****
20 REM *** CREATES OPTIMIZED CODE FOR 'SBLOCK' PROGRAM ***
30 CLS:PRINTTAB(11)"STRING  BLOCK  BUILDER  PROGRAM":PRINTTAB(17)"BY J
    EFF W. COLLINS":PRINT"ASSEMBLY PROGRAM 'SBLOCK' MUST BE RESIDENT.
    ":INPUT"WHEN READY FOR INSTRUCTIONS, HIT <ENTER>";I
40 CLS:CLEAR1000:DEFINT C,I,J,S,Z
50 PRINT"IF THE STRING BLOCK TO BE CREATED IS TO BE PRINTED OVER SOME-
    ":PRINT"THING ELSE, DO YOU WANT IT TO HAVE:  0) NON-DESTRUCTIVE S
    PACES"
60 PRINTTAB(37)"1) DESTRUCTIVE SPACES"
70 INPUT"CHOOSE (0 OR 1)";SP:IFSP>1 OR SP<0 THEN 70
80 CLS:IF SP <> 0 THEN SP=32
90 PRINTTAB(4)"KEY";:PRINTTAB(25)"ACTION":PRINT"RIGHT ARROW   --  SET
    COLUMN X AT ROW Y."
100 PRINT"UP ARROW   --   CHOICE OF ALPHANUMERICS (AT CURRENT LOC'N)":
    PRINTTAB(16)"OR GRAPHICS STARTING AT ROW 0, COLUMN 0."
110 PRINT"<ENTER>   --";:PRINTTAB(16)"FINISH ALPHA OR GRAPHIC LINE. IF
     GRAPHIC LINE,":PRINTTAB(16)"RESUME AT NEXT HIGHER LINE. IF FROM
    ALPHA LINE":PRINTTAB(16)"RESUME GRAPHIC MODE AT CURRENT LOCATION.
    "
130 PRINT"LEFT ARROW  --";:PRINTTAB(16)"FROM GRAPHICS MODE ONLY. BACKS
    PACE ONCE TO":PRINTTAB(16)"DELETE A GRAPHICS CHARACTER, TWICE TO
    DELETE AN":PRINTTAB(16)"ALPHA CHARACTER, RESUMING AT DELETED CHAR
    ACTER'S";:PRINTTAB(16)"LOCATION."
140 PRINT"AMPERSAND  --";:PRINTTAB(16)"FROM GRAPHICS MODE ONLY. PROCES
    S THE BLOCK":PRINTTAB(16)"STRING, SHOW ITS BYTES, OFFER TO RECORD
     ON TAPE.";
150 PRINT"SPACE BAR  --";:PRINTTAB(16)"SKIP ONE GRAPHICS OR ONE CHARAC
    TER SPACE.":PRINT"T --";:PRINTTAB(16)"READ IN TAPE BLOCK STRING."
    :INPUT"HIT 'ENTER' TO START.";I:CLS
```

Program continued

```
160 P=805:BL$=STRING$(19,32):G$="*  GRAPHICS MODE  *"
170 PRINT@P,G$;:GOSUB 250
180 KP$="":KP$=INKEY$:IF KP$="" THEN 180 ELSE K=ASC(KP$):IF K=8 AND X
     <> 0 THEN X=X-1:RESET(X,Y):GOSUB 250:GOTO 180
185 IF K=84 THEN PRINT@P,BL$;:PRINT@P,"*  TAPE READY <EN>  *";:B$="":B
    $=INKEY$:IFB$=""THEN 185 ELSE IF B$ <> CHR$(13) THEN 170 ELSE INP
    UT#-1,A$:CC=0:GOSUB 800:PRINT@0,"";:Z=USR(1):Z=USR(VARPTR(A$)):A$
    ="":GOTO 170
190 IF K=32 THEN X=X+1:GOSUB 320:GOTO 180
200 IF K=13 THEN X=0:Y=Y+1:GOSUB 320:GOTO 180
210 IF K=9 AND X < 128 AND Y < 48 THEN SET(X,Y):X=X+1:GOSUB 320:GOTO 1
    80
220 IF K=38 THEN 350
230 IF K=91 THEN 260
240 GOSUB 250:GOTO 180
250 PRINT@875,"Y ROW=";Y;" ";:PRINT@939,"X COLUMN=";X;" ";:RETURN
260 PRINT@P,BL$;:PRINT@P,"(G)RAPHICS (A)LPHA?";:B$="":B$=INKEY$:IF B$=
    "" THEN 260 ELSE IF NOT (B$="G" OR B$="A") THEN 260 ELSE PRINT@P,
    BL$;:IF B$="G" THEN X=0:Y=0:PRINT@P,G$:GOSUB 250:GOTO 180 ELSE IF
     B$="A" THEN PRINT@P,"  *  ALPHA MODE  *";
270 TY=Y/3:TY=INT(TY)
280 TX=X/2:IF TX <> INT(TX) THEN TX=TX+1
290 J=0
300 B$="":B$=INKEY$:IF B$="" THEN 300 ELSE B=ASC(B$):IF B=8 THEN B=93
    ELSE IF B=9 THEN B=94 ELSE IF B=10 THEN B=92 ELSE IF B=13 THEN PR
    INT@P,G$;:GOSUB 250:GOTO 180
310 PRINT@(64*TY)+TX+J,CHR$(B);:J=J+1:X=X+2:GOSUB 320:GOTO300
320 IF X > 127 THEN X=0 ELSE IF X > HX THEN HX=X
330 IF Y > 47 THEN Y=0:X=0 ELSE IF Y > HY THEN HY=Y
340 GOSUB 250:RETURN
350 PRINT@P,BL$;:PRINT@P,"*  PROCESSING ...";:FOR J=1 TO 100:NEXT J
360 CA=15360:I=0
370 X=HX/2:IF X <> INT(X) THEN X=X+1
380 Y=HY/3:Y=INT(Y)
390 T1=-1:T2=1
400 FOR J=0 TO X-1
410 CP=PEEK(CA+J)
420 IF CP=32 OR CP=128 THEN CP=SP
430 IF CP=T1 THEN T2=T2+1:IF T2=256 THEN A$=A$+CHR$(128)+CHR$(255)+CHR
    $(CP):T2=1:GOTO 450 ELSE 450
440 IF T1<>-1 THEN IF T2=2 THEN A$=A$+CHR$(T1)+CHR$(T1):T2=1 ELSE IF T
    2>2 THEN A$=A$+CHR$(128)+CHR$(T2)+CHR$(T1):T2=1 ELSE A$=A$+CHR$(T
    1)
450 T1=CP
460 NEXT J
470 CA=CA+64
480 I=I+1
490 IF I>Y THEN 510
500 GOTO 400
510 IF T2>1 THEN A$=A$+CHR$(128)+CHR$(T2)+CHR$(T1) ELSE A$=A$+CHR$(CP)

515 IF LEN(A$) > 2 THEN IF MID$(A$,LEN(A$)-2,1)=CHR$(128) AND MID$(A$,
    LEN(A$),1)=CHR$(0) THEN A$=LEFT$(A$,LEN(A$)-3) ELSE IF MID$(A$,LE
    N(A$),1)=CHR$(0) THEN A$=LEFT$(A$,LEN(A$)-1)
520 Z=LEN(A$)+1
530 A$=CHR$(X)+A$:IF X=0 THEN A$="":GOTO 170
540 CLS:PRINT"THE LENGTH OF THIS BLOCK STRING IS";STR$(Z);"."
550 IF Z >248 THEN PRINT"**** WARNING **** THIS STRING IS TOO LONG":PR
    INT"TO GO TO TAPE AS IS. REWORK THE GRAPHIC."
560 B$="":IF Z < 249 THEN PRINTTAB(5);"DO YOU WANT TO SAVE THE STRING
    ON TAPE (Y/N)";:INPUT B$:IF LEFT$(B$,1)="Y" THEN INPUT"HIT 'ENTER
    ' WHEN TAPE IS READY.";B$:PRINT#-1,A$
570 PRINT"THE STRING'S DECIMAL BYTES WILL FOLLOW."
580 INPUT"HIT 'ENTER' WHEN READY.";I
590 FOR I=1 TO LEN(A$)
600 PRINT ASC(MID$(A$,I,1));:FOR J=1 TO 25:NEXT J
610 NEXT I
620 PRINT:PRINT"THERE ARE";Z;"BYTES (INCLUDES COLUMN COUNT) IN THIS ST
    RING."
630 T1=HY:PRINT"TO CHANGE HIGHEST ROW Y,";STR$(HY);", ";:INPUT"ENTER 0
```

```
    -47 ELSE HIT <ENTER>";HY
640 T2=HX:PRINT"TO CHANGE HIGHEST COLUMN X,";STR$(HX);", ";:INPUT"ENTE
    R 0-127 ELSE HIT <ENTER>";HX:IF T1 <> HY OR T2 <> HX THEN CLS:Z=U
    SR(VARPTR(A$)):A$="":GOTO 350
650 INPUT"HIT 'ENTER' TO DISPLAY STRING AND CONTINUE";Z
660 CLS
670 Z=USR(1)
680 Z=USR(VARPTR(A$))
690 X=0:Y=0:A$=""
700 GOTO 170
800 FOR Z=2 TO LEN(A$)
810 IF MID$(A$,Z,1)=CHR$(128) THEN CC=CC+ASC(MID$(A$,Z+1,1))-3
820 CC=CC+1:NEXT Z
830 HY=CC/ASC(A$):IF HY=INT(HY) THEN HY=HY-1 ELSE HY=INT(HY)
840 HY=(HY*2)+HY+2:HX=(ASC(A$)*2)-1:X=HX:Y=HY:RETURN
```

Program Listing 3. *Demonstration of screen printing*

```
10 GOTO 30
20 X=USR(US):X=USR(VARPTR(A$(SI))):RETURN
30 CLS:PRINT"PROGRAM TO DEMONSTRATE 'SBLOCK' ASSEMBLY PROGRAM,"
40 PRINT"SHOWING THE EFFECT OF OVERLAYING IMAGES USING"
50 PRINT"DESTRUCTIVE AND NON-DESTRUCTIVE SPACES."
60 PRINT:PRINT"IF A/L 'SBLOCK' IS IN PROTECTED MEMORY,"
70 PRINT"PRESS 'ENTER' FOR INSTRUCTIONS.":GOSUB 550
80 CLS:PRINT"KEY";TAB(16)"ACTION":PRINT
90 PRINT"D  --";TAB(16)"DISPLAY CURRENT STRING BLOCK AT CURSOR"
100 PRINT"N  --";TAB(16)"DISPLAY NEXT STRING BLOCK"
110 PRINT"E  --";TAB(16)"ERASE CURRENT STRING BLOCK"
120 PRINT"SPACE BAR  --";TAB(16)"ADVANCE CURSOR"
130 PRINT"LEFT ARROW  --";TAB(16)"BACKSPACE CURSOR"
140 PRINT"RIGHT ARROW  --";TAB(16)"ADVANCE CURSOR 8 SPACES"
150 PRINT"DOWN ARROW  --";TAB(16)"SKIP CURSOR DOWN A LINE"
160 PRINT"UP ARROW  --";TAB(16)"SKIP CURSOR UP A LINE"
170 PRINT"C  --";TAB(16)"CLEARS SCREEN, KEEPS SAME CURSOR LOCATION"
180 CLEAR 1000
190 DEFINT B-Z
200 DEFSTR A,C,K
210 K$(1)="GOODBYE":K$(2)="BLUB...BLUB..."
220 B=13:SS=15360:DIM A$(B)
230 FOR I=0 TO B
240 READ NB:FOR J=1 TO NB:READ SB:A$(I)=A$(I)+CHR$(SB):NEXT J
250 NEXT I
260 PRINT:PRINT"WHEN READY TO START HIT 'ENTER'.":GOSUB 550
270 CLS:PRINT@452,"THE CURSOR COMING UP IS YOUR PROMPT."
280 FOR I=0 TO 1023 STEP 12
290 PRINT@I,"";:US=1:SI=9:GOSUB 20 :PRINT@1023-I,"";:GOSUB 20
300 PRINT@1023-I,"";:US=0:GOSUB 20:PRINT@I,"";:GOSUB 20
310 NEXT I:CLS
320 CU$=CHR$(1)+CHR$(95)
330 VI=0:IX=0
340 PA=SS
350 PV=PEEK(PA)
360 PRINT @ VI,"";:X=USR(1):X=USR(VARPTR(CU$))
370 KP$=INKEY$:IF KP$=""THEN 370
380 IF KP$="D" THEN PRINT@PA-SS,CHR$(PV);:US=1:SI=IX:GOSUB 20:PV=PEEK(
    VI+SS):GOTO 370
390 IF KP$="E" THEN PRINT@PA-SS,CHR$(PV);:US=0:SI=IX:GOSUB 20:PV=PEEK(
    VI+SS):GOTO 370
400 IF KP$="N" THEN IX=IX+1:KP$="D":IF IX > B THEN 560 ELSE 380
410 IF KP$="C" THEN PV=32:CLS:GOTO 360
420 IF KP$=CHR$(9) THEN NA=VI+8:GOTO 470
430 IF KP$=CHR$(8) THEN NA=VI-1:GOTO 470
440 IF KP$=CHR$(10) THEN NA=VI+64:GOTO 470
450 IF KP$=CHR$(32) THEN NA=VI+1:GOTO 470
460 IF KP$=CHR$(91) THEN NA=VI-64 ELSE GOTO 370
470 IF NA > 1023 THEN NA=1023
```

Program continued

```
480 IF NA < 0 THEN NA=0
490 PRINT@VI,"";:X=USR(0):X=USR(VARPTR(CU$))
500 PRINT@PA-SS,CHR$(PV);
510 PA=NA+SS
520 PV=PEEK(PA)
530 VI=NA
540 GOTO 360
550 KP$="":KP$="":KP$=INKEY$:IF KP$="" THEN 550 ELSE RETURN
560 CLS:FOR I=0 TO 20:PRINT@I,"";:US=1:SI=12:GOSUB 20:FOR Z=1 TO 50:NE
    XT Z:US=0:GOSUB 20:NEXT I
570 FOR I=1 TO 14:J=J+64:PRINT@J,"";:US=1:GOSUB 20:FOR Z=1 TO 500:NEXT
      Z:US=0:GOSUB 20:PRINT@J+30,K$(RND(2));
580 FOR Z=1 TO 39:NEXT Z
590 NEXT I:FOR I=1 TO 1000:NEXT I:END
600 X=USR(US):X=USR(VARPTR(A$(SU))):RETURN
610 DATA 4,1,191,143,131
620 DATA 4,3,191,0,191
630 DATA 15,5,176,191,176,191,176,140,191,140,191,140,0,131,0,131
640 DATA 14,5,176,140,191,140,140,176,179,191,179,140,0,0,131
650 DATA 17,5,0,143,143,176,140,176,140,131,176,176,128,3,0,128,2,131
660 DATA 16,5,188,131,188,0,0,188,131,140,176,140,0,131,131,0,131
670 DATA 5,3,0,191,143,131
680 DATA 10,3,176,140,131,143,176,128,3,0,131
690 DATA 8,3,131,140,176,0,176,143,131
700 DATA 14,5,140,176,191,176,140,179,143,191,143,179,0,0,131
710 DATA 9,5,176,176,191,176,176,0,0,143
720 DATA 20,11,131,140,179,140,179,140,179,140,179,140,131,128,3,0,131
    ,140,179,140,131
730 DATA 100,29,128,12,0,176,184,188,191,188,180,176,144,128,10
740 DATA 0,128,4,176,128,5,0,136,128,10,191,189,188,180,176
750 DATA 176,144,0,0,131,128,5,191,189,188,188,128,12,191
760 DATA 176,176,191,191,128,3,143,133,139,128,3,143,131,131
770 DATA 128,3,0,143,175,128,11,191,143,135,128,4,131,129,78
780 DATA 79,78,0,68,69,83,84,128,3,0,139,175,159,128,4,143
790 DATA 191,159,143,129,128,19,0,130,128,5,0,130
800 DATA 76,20,143,143,128,4,32,143,143,128,4,32,143,143,128,4,32
810 DATA 143,143,68,69,83,84,82,85,67,84,73,86,69,32,83
820 DATA 80,65,67,69,83,32,32,128,20,140,176,176,128,4,32
830 DATA 176,176,128,4,32,176,176,128,4,32,176,176,131,131,128
840 DATA 4,32,131,131,128,4,32,131,131,128,4,32,128,2,131
```

HARDWARE

HEART/BAS HEART/CIM

HEART/BAS HEART/CIM

by Alan Sehmer

My doctor recommended that I peddle an exercycle for a total of 30 minutes. During the first five minutes, I have to bring my heart rate up to 150 beats per minute. I must hold that rate for 20 minutes. The last five minutes are used to cool off and bring the rate back down to normal. After the first week of this exercise, I learned that the combination of riding the exercycle and monitoring my heart rate was a boring form of torture.

With my TRS-80 Model I, I can monitor my heart rate. The data can be handled and displayed in many ways. The first is to display the heart rate on the CRT. I can also get a hard-copy record which enables me to look for any trends over time. With a Line Printer II, the heart rate and time can be printed in one of two forms—as a table or, with a little more programming, as a graph. If you don't have a printer, you can send the data to the CRT as a table or graph or store it on a tape or disk. The two main parts of this project are the EKG hardware and the machine-language driver program. Once you have these, what you do with the data is up to you.

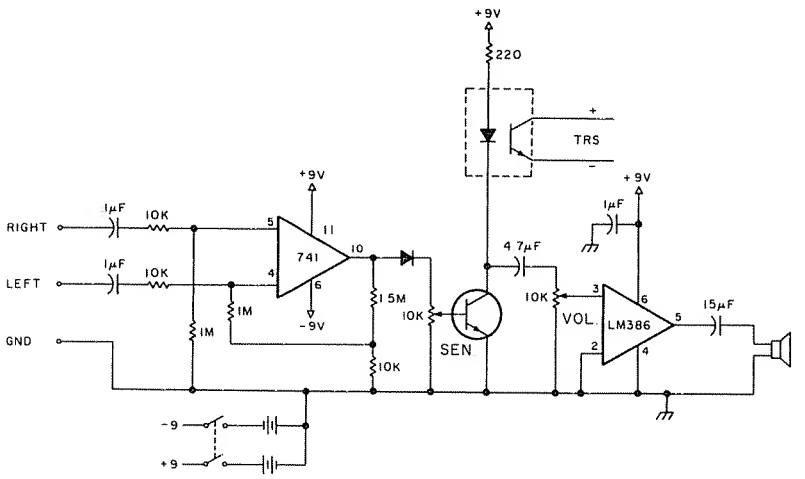


Figure 1. *EKG*

For this project, you need about $25.00, two evenings, a Model I TRS-80 with one bit of input port, and the Expansion Interface with disk (the program uses the clock). The schematic for the EKG is given in Figure 1. The unmarked diode and transistor can be almost anything in your junk box. All parts are from Radio Shack. The only construction caution is to be sure not to connect the EKG ground to TRS ground (opto-isolator negative). If grounds are connected, the TRS digital noise interferes with the EKG. The opto-isolator also electrically disconnects you from the computer; connecting grounds would defeat this. For the electrodes, I used the metal caps from the cans that 35mm photographic film comes in. Photos 1 and 2 show the internal layout of my EKG; photo 3 is the completed unit with its film can electrodes.

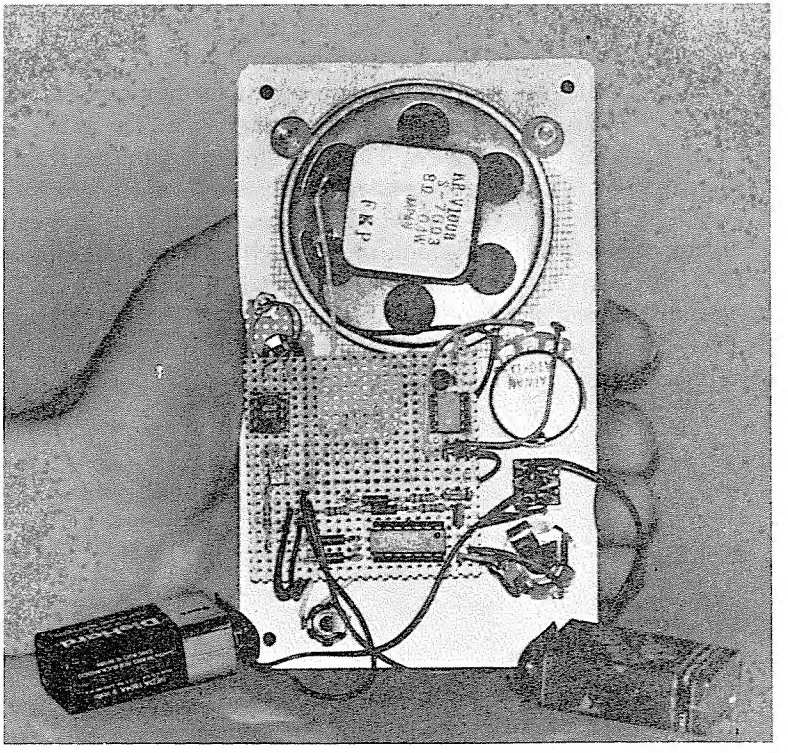


Photo 1. *Internal layout of EKG*

To use the EKG, place the left and right electrodes on the chest about two inches above each nipple and place the ground electrode on the waist toward the right. The ground electrode is held in place by the waistband of my slacks, and the chest electrodes are held by an ace bandage. I have found

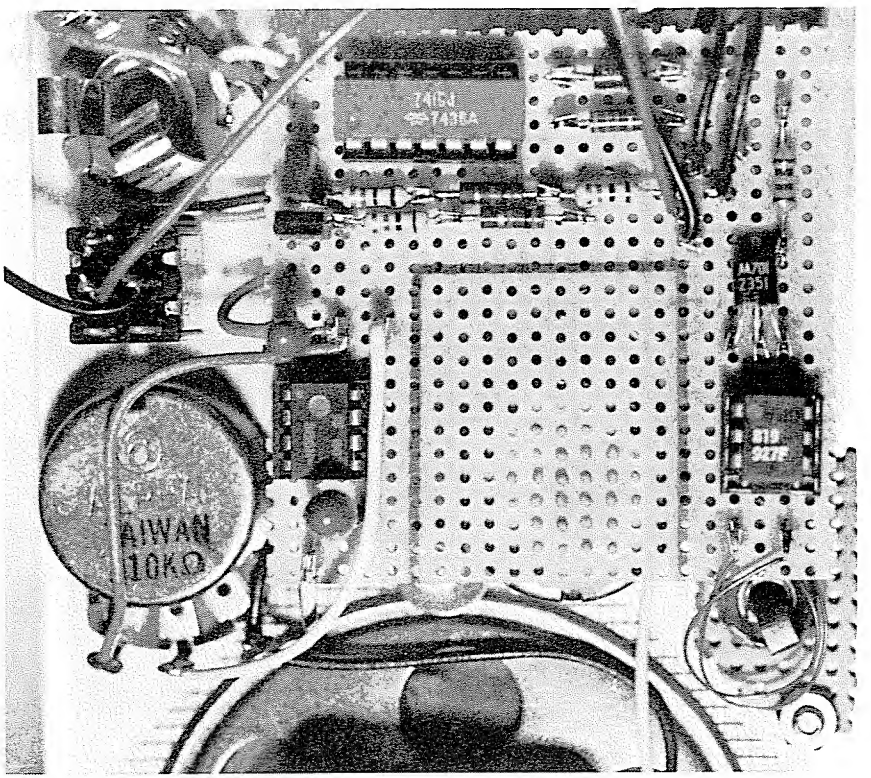


Photo 2. *Internal layout of EKG*

it easier to wrap my chest with the bandage first and then position the electrodes. Some sort of conductive cream should be used between the skin and electrode. You can obtain this from any surgical supply house. Once you are wired for sound, plug yourself in and turn on the EKG. Turn the volume up about half way and then slowly turn up the sensitivity. About half way up you should hear your heart beat; if you turn it up more the heart beat will be lost in noise. The EKG does not listen for the sound of the heart but is sensitive to the electrical signal of the heart muscle. It is also sensitive to any other electrical signal in the chest area. Place the palms of your hands together and push. You should hear the signals of the chest muscles. The EKG is safe to use because it is battery powered and the computer output goes through an opto-isolator. To stay on the safe side, do not replace the batteries with battery eliminators.

When the machine-language driver program (see Program Listing 1) is called, it first sets the heart beat counter to zero, then keeps checking the clock, waiting for the seconds to change. When they do, the program starts

checking for the EKG (I used port #4) and clock signals. When the clock has advanced 20 seconds, the program returns with the number of heart beats in those 20 seconds.

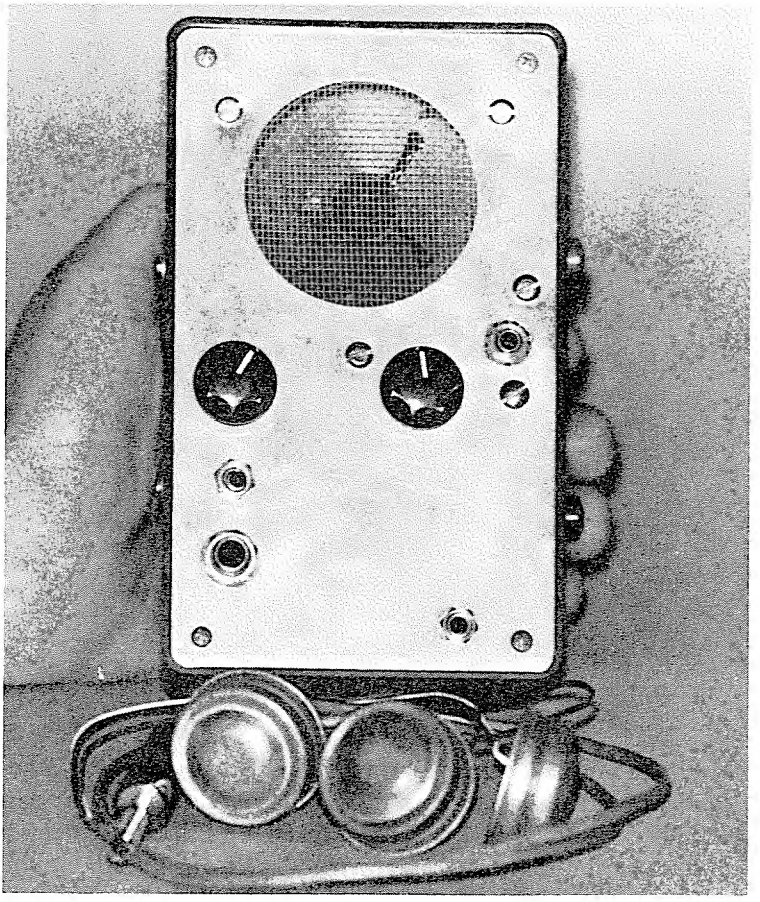


Photo 3. *The completed unit with its film can electrodes*

Once the numbers are in the computer, the BASIC program (see Program Listing 2) outputs the data. Every 20 seconds, this program displays the elapsed time and current heart rate. It also stores the heart data in array Y. I decided to monitor my heart for 40 minutes instead of only 30 minutes. During the last 10 minutes, I get off the exercycle and sit quietly. This allows my heart to come down completely to its resting rate. The doctor didn't request this, but I thought it was a nice touch. After 40 minutes, the program plots heart rate versus time. (See Figure 2.) The plot section of the program is for the Line Printer II, but you can modify it for use with another printer.

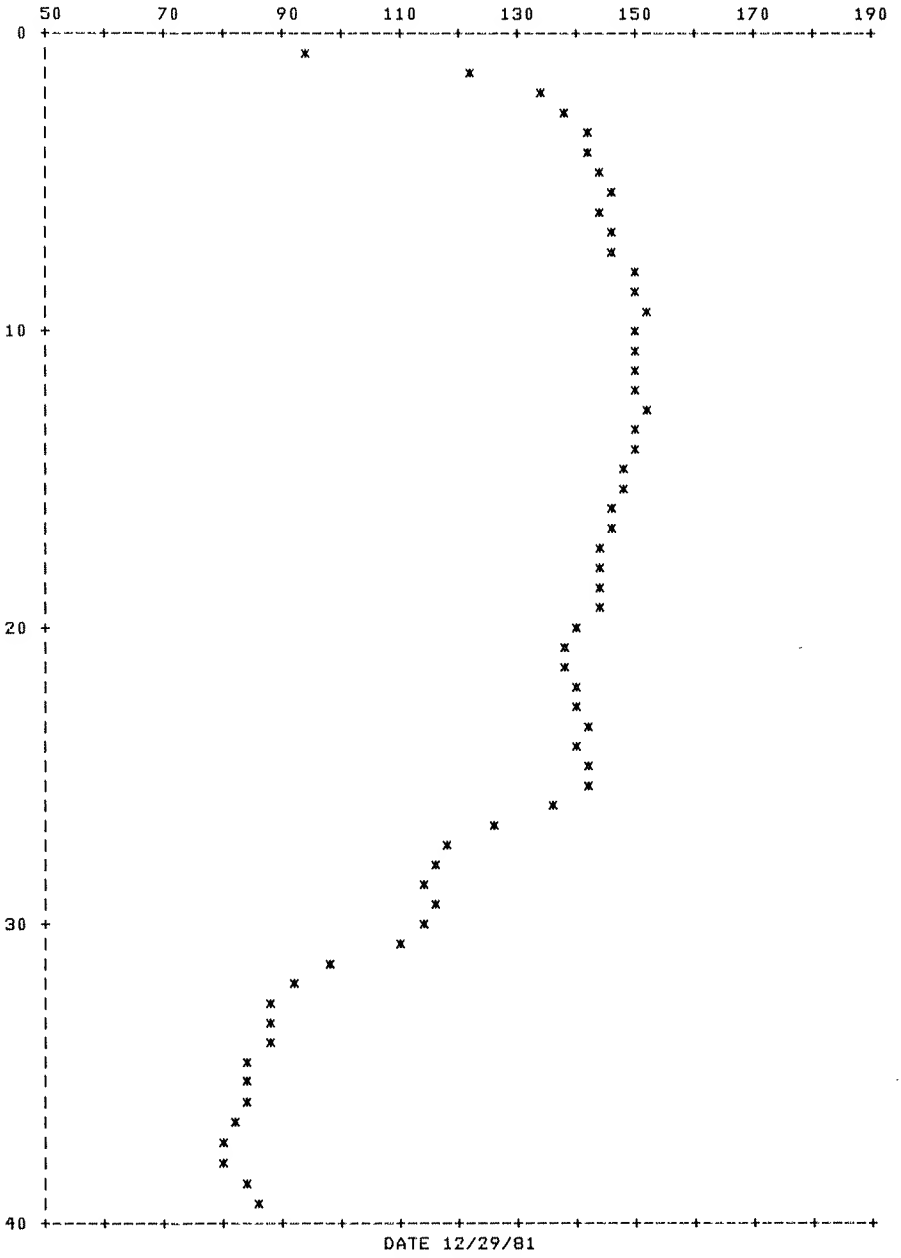


Figure 2. *Plot of heart rate versus time*

One last word of caution: The numbers for heart rates and times were given to me by my doctor. This type of exercise program should not be a do-it-yourself project. Go to your doctor and get your own numbers.

Program Listing 1. *Assembly-language program*

```
            00100 ;       HEART BEAT     CLOCK MUST BE ENABLED
A000        00110         ORG     0A000H
A000 110000 00120         LD      DE,0000H        ;INIT BEAT COUNT
A003 0614   00130         LD      B,20            ;INIT # OF SECONDS
A005 214140 00140         LD      HL,4041H        ;PT. TO TRS CLOCK
A008 7E     00150         LD      A,(HL)          ;WAIT FOR SEC. TO CHANGE
A009 BE     00160 WAIT    CP      (HL)
A00A 28FD   00170         JR      Z,WAIT
A00C 7E     00180 STIME   LD      A,(HL)          ;STORE NEW TIME
A00D 323CA0 00190         LD      (TIME),A
A010 DB04   00200 INPUT   IN      A,(4)           ;USES PORT #4
A012 FEFF   00210         CP      0FFH
A014 2008   00220         JR      NZ,PULSE
A016 3A3CA0 00230 CTIME   LD      A,(TIME)        ;HAVE SECONDS CHANGED ?
A019 BE     00240         CP      (HL)
A01A 201A   00250         JR      NZ,TEST         ;NZ=HAVE CHANGED.
A01C 18F2   00260         JR      INPUT
A01E 13     00270 PULSE   INC     DE              ;BUMP BEAT COUNT
A01F 3E2A   00280         LD      A,2AH
A021 320A3C 00290         LD      (3C0AH),A
A024 210040 00300         LD      HL,4000H        ;100MS DELAY FOR PULSE
A027 2B     00310 DELAY   DEC     HL              ;  TO GO AWAY
A028 7C     00320         LD      A,H
A029 B5     00330         OR      L
A02A 20FB   00340         JR      NZ,DELAY
A02C 3E20   00350         LD      A,20H
A02E 320A3C 00360         LD      (3C0AH),A
A031 214140 00370         LD      HL,4041H        ;RESTORE CLOCK POINTER
A034 18E0   00380         JR      CTIME           ;GO CHECK TIME
A036 10D4   00390 TEST    DJNZ    STIME           ;DEC SECONDS COUNT
A038 EB     00400         EX      DE,HL           ;BEAT CT. TO HL FOR BASIC
A039 C39A0A 00410         JP      0A9AH           ;BACK TO BASIC
A03C 00     00420 TIME    DEFB    00
0000        00430         END
00000 TOTAL ERRORS
```

Program Listing 2. *BASIC program*

```
10 DEFUSR 3=(&HA000) : CLEAR 100 : DIM Y(120)
20 INPUT "PRESS ENTER TO START";A$ : CLS
30 PRINT"TEST" : Z=USR 3(0) : CLS : PRINT CHR$(23)
40 FOR Z=1 TO 120 : Y=USR 3(0)*3 : PRINT@ 0,Y
50 PRINT@ 64,INT(Z/3);"MINUTES"; : Y(Z)=Y-50
60 NEXT : CLS : PRINT@ 128,"DONE"
70 PRINT "BACKUP PRINTER PAPER TWO LINES FROM CUTTER BAR"
80 LINEINPUT "ENTER DATE ";D$
90 LPRINT TAB(5)"50"TAB(15)"70"TAB(25)"90 ";
100 FOR X=110 TO 190 STEP 20 : LPRINT STRING$(5," ");X; : NEXT
110 LPRINT "     0 +"; : FOR X=1 TO 7 : LPRINT "----+----+"; : NEXT :
    LPRINT ""
120 FOR Z=2 TO 118 STEP 2
130 IF Z/30=INT(Z/30) LPRINT " ";Z/3;"+"; : GOTO 150
140 LPRINT TAB(5)CHR$(124);
150 IF Y(Z)<0 OR Y(Z)>140 LPRINT "" : GOTO 170
160 LPRINT STRING$(Y(Z)/2-1," ");"*"
170 NEXT
180 LPRINT "  40 +"; : FOR X=1 TO 7 : LPRINT "----+----+"; : NEXT : LP
    RINT ""
190 LPRINT TAB(34)"DATE ";D$ : END
200 DEFUSR 3=(&HA000) : CLS
210 PRINT@ 0,USR 3(0)*3 : GOTO 210
220 0=USR 1(Y) : 0=USR 2(X) : RETURN
```

HOME APPLICATIONS

Low Resolution Voice for the Color Computer
Planning Your Retirement

Low Resolution Voice for the Color Computer

by Dr. Edward Kimble

With a host of articles available concerning voice synthesis and recording using more expensive systems, it is pleasing to find that Radio Shack's Color Computer is capable of both low and high resolution speech manipulation.

While it may be somewhat crude, the simplest method for entering speech into the Color Computer is to use the cassette input to measure the zero crossing of the audio signal. When you work with this type of input, you'll find that the electrical signal is not capacitively coupled to the input. This means that the bias and impedances can be externally adjusted, which minimizes the noise that might result from spurious zero crossings. With cassette input, you may also hook up different tape recorders to the input to generate different sound characteristics. Using the six Program Listings, you can feed audio information into the computer by either using the cassette recorder in the play mode or by speaking into the microphone while the recorder is in the record mode. This method can be dangerous if you accidentally leave a valuable recording in the slot. In either case you would be wise to adjust your recorder to reduce the noise level before you start. Do this by adjusting the record level, treble, and bass. If you have a choice of recorders, try them all, because one may work better than another.

There are hundreds of ways to encode speech and place it in memory. I have chosen two techniques. One method feeds the waveform into memory while displaying the impulses directly on the high-resolution screen. (See Program Listings 1–3.) My daughter has found this to be useful when practicing her music lessons, since small changes in frequency or accent show up as large changes on the display.

The audio quality can be improved somewhat in these programs if you increase the sampling rate. Quality is limited when the smoothly-varying waveform is converted into a square wave by the zero crossing detector circuitry. Despite these problems, when the data is played back through the single-bit sound output, individual voices are clearly recognized and understood. At the end of the first driver program (Program Listing 3) is a graphics routine that you can gain access to by typing GOTO 89 (ENTER). Since the data is stored as spatial relationships on the screen, unusual figures plotted on the screen often come out as unusual sounds when the playback routine is executed.

An alternate technique that is often used to record sound stores the pulse widths of the incoming waves in memory as discrete numbers. This method can be inefficient in that it requires a single byte of memory for each half wave of audio information. On the other hand, it is efficient in storing pauses and periods of low frequency. For example, the letter s contains enough high-frequency variations that memory may be filled by a word like Mississippi. Program Listings 4–6 use this method. As you run the programs, note that it is much more difficult to determine from the display screen whether you have recorded information or noise. One advantage this technique offers is that a crude voice print, similar to those used in forensic science, can be quickly prepared. The pulse widths are plotted versus time.

If you run the second series of programs (Program Listings 4–6), you will see why I plotted the dependent variable on the horizontal axis. In preparing the voice print, the pulse width data is erased starting at the beginning of the data in the upper right-hand corner of the screen. Replacing this data are dots whose distance from the right-hand edge of the screen is determined by the magnitude of the pulse widths. You can examine the plot and determine how many waves of a given pulse width were present at a given time.

	Program set 1	Program set 2
Recording speed	3025 MSB & 3026 LSB	3215 & 3216
Playback speed	321C & 321D	3033 & 3034
Start of record	3010 & 3011	3205 & 3206
storage	3201 & 3202	3010 & 3011
End of record	3042 & 3043	3234 & 3235
	3236 & 3237	3027 & 3028
Page address	3015, 3204	3201

Table 1. *Some useful addresses (hexadecimal)*

To enter the machine-language programs, you must have a monitor or a short BASIC program that POKEs the instructions into memory. I've included a short program in BASIC which acts as a driver program, turning on and off the machine-language routines. For wave plotting program number one, the storage routine for recording your voice begins at hexadecimal location 3000 (decimal location 12288). Each of the routines begins execution with the first byte. You can EXEC these routines without the use of the driver program; it is merely added as a convenience. The machine-language programs are listed here in blocks of eight bytes with the address of the first byte in each block given at the left. These programs are relocatable; however, there is a memory location used for temporary storage at address 3250 in hexadecimal. The first two most significant digits of this address can

be changed so as to store the information on a different page of memory by going to the Page Address as listed in Table 1 and changing the numbers at these addresses to some number other than 32. Be sure you have RAM storage at the new location. To decrease recording speed, you can increase the numbers for the recording speed and playback speed locations given in Table 1. Filtering and frequency shifting is done, as indicated at the end of driver program number two, by using IF-THEN statements to sample the data and render a new frequency or response.

Although the tone quality of this system is not remarkable, the abilities exhibited by these two methods are quite remarkable. Up to 15 seconds of speech can be recorded in as little as five kilobytes. You can do frequency analysis studies that were previously limited to the more sophisticated college labs. Deaf students can utilize this technique to see the vibrations in their voices as well as such properties as frequency and attack without the aid of a storage oscilloscope.

Program Listing 1. *Wave recording program*

```
WAVE RECORDING PROGRAM

3000  86 38 B7 FF 21 86 FE B7
3008  FF 20 86 3C B7 FF 21 8E
3010  06 00 C6 00 86 32 1E 8B
3018  0F 50 86 01 B4 FF 20 9A
3020  50 97 50 5C CE 00 02 33
3028  5F 11 83 00 00 27 02 20
3030  F6 C1 08 27 04 08 50 20
3038  E1 A7 84 C6 00 0F 50 30
3040  01 8C 1D B0 27 02 20 D2
3048  86 00 1E 8B 39 00 FF 00
```

Program Listing 2. *Playback program*

```
PLAYBACK PROGRAM

31E5  C6 00 86 33 B7 FF 23 86
31ED  FF B7 FF 22 86 37 B7 FF
31F5  23 86 35 B7 FF 03 86 B4
31FD  B7 FF 01 8E 06 00 86 32
3205  1E 8B A6 84 97 50 09 50
320D  25 07 86 FF B7 FF 22 20
3215  05 86 FD B7 FF 22 CE 00
321D  02 33 5F 11 83 00 00 27
3225  02 20 F6 5C C1 08 27 02
322D  20 DC C6 00 12 12 30 01
3235  8C 1D B0 27 02 20 CB 86
323D  00 1E 8B 39 00 FF 00 FF
```

Program Listing 3. *Driver program*

```
DRIVER PROGRAM

5 CLS
10 INPUT "CHOOSE PLAY'P' OR RECORD'R'";D$:IF D$="P" THEN 60
20 PMODE 4,1:PCLS:SCREEN1,1
30 EXEC12288
40 IF INKEY$="" THEN 40
50 GO TO 5
60 PMODE 4,1:SCREEN1,1:EXEC12773
70 PRINT "DONE"
80 GO TO 5
89 PMODE 4,1:PCLS:SCREEN1,1
90 FOR A=0 TO 162 STEP 2
91 LINE (0,A)-(COS(A/30)*A*1.2+160,A+26),PSET:NEXT
```

Program Listing 4. *Pulse width encoding program*

```
PULSE WIDTH ENCODING PROGRAM

3200  86 32 1E 8B 8E 06 00 C6
3208  00 B6 FF 20 84 01 D7 50
3210  91 50 26 1B CE 00 01 33
3218  5F 11 83 00 00 27 02 20
3220  F6 6C 84 97 50 86 FF A1
3228  84 27 06 96 50 20 DA 1E
3230  89 30 01 8C 1D B0 27 02
3238  20 CF 86 00 1E 8B 39 00
```

Program Listing 5. *Pulse width playback program*

```
PULSE WIDTH PLAYBACK PROGRAM

3000  86 33 B7 FF 23 86 FF B7
3008  FF 22 86 37 B7 FF 23 8E
3010  06 00 86 FF 20 02 86 FD
3018  B7 3E 80 E6 84 C1 FF 27
3020  03 B7 FF 22 30 01 8C 1D
3028  B0 27 1F A6 84 81 00 27
3030  10 4A CE 00 03 33 5F 11
3038  83 00 00 27 02 20 F6 20
3040  EC B6 3E 80 81 FD 27 CA
3048  20 CC 39 00 FF 00 FF 00
```

Program Listing 6. *Pulse width driver program*

```
PULSE WIDTH DRIVER PROGRAM

10 DIM D(35)
20 CLS:INPUT"RECORD='R',PLAY='P',SPECTRUM='S'";S$:IF S$="R" THEN 50 ELSE IF S$="
P" THEN 30 ELSE 60
30 PMODE 4,1:SCREEN1,1:EXEC(12288)
40 IF INKEY$="" THEN 40 ELSE 20
50 PMODE 4,1:SCREEN1,1:PCLS:EXEC(12800):GO TO 20
60 PMODE 4,1:SCREEN1,1:K=-1
70 FOR A=1536 TO 7600 STEP 32
80 K=K+1
90 FOR B=0 TO 31
100 D(B)=PEEK(A+B)
110 NEXT B
120 FOR W=1 TO 255:PRESET(W,K):NEXT W
130 FOR C=0 TO 31
140 PSET(D(C),K)
150 NEXTC
160 NEXT
170 GO TO 20
180 FOR A=1536 TO 7600
190 F=PEEK(A)
200 IF F<3 THEN POKE A,0
210 PRINT PEEK(A)
220 NEXT
```

Planning Your Retirement

by R. L. Conhaim

With all the talk of Social Security cutbacks, and with the many unknowns about job retirement plans, young people today look to their retirement as an unknown quantity. There are so many variables that no young person can say with any degree of accuracy how much retirement income will be available. To rely totally on Social Security and job retirement income may be dangerous. About the only sensible thing to do is to establish a supplemental retirement plan, putting aside a sum each year. The government has encouraged such supplementary savings through Individual Retirement Accounts (IRA) and Keogh plans for the self-employed. How much to put aside, and how much retirement income it will produce is usually a matter of guesswork. By using well known annuity formulas, it is possible to forecast accurately how much will be available and how much income that principal will produce.

The Retirement program (see Program Listing) shows how far savings will go, or establishes a savings goal if a monthly supplemental retirement income is specified. Assuming you know how much supplemental retirement income you want, for how long, and at what interest rate, the program will show how much annual savings are needed now. It does this in two steps. First, it computes the total principal needed at retirement, using the annuity formula:

$$P = A \frac{(1 + I_{eff})^n - 1}{(1 + I_{eff})^n \times I_{eff}}$$

where:

P = Principal required
A = Amount of monthly annuity
I_{eff} = Effective annual interest rate
n = Total number of years annuity is to run
S = Amount of annuity savings required

The program then computes the annual savings needed to provide the required principal, using the Sinking Fund formula:

$$S = \frac{P \times I_{eff}}{(1 + I_{eff})^{n1} - 1}$$

The variable n1 is the number of years to retirement.

If you would rather specify how much you can set aside each year for your retirement, the program will compute how much monthly retirement you

will receive, and for how long. It does this with two additional formulas. First, it computes the principal your savings will generate, using the formula:

$$P = S \frac{(1 + I_{eff})^{nl} - 1}{I_{eff}}$$

Then, it computes the monthly annuity that the principal will provide for the specified number of years, using the formula:

$$A = \frac{P \times I_{eff} \times (1 + I_{eff})^{n}}{(1 + I_{eff})^{n} - 1}$$

Since you are computing *annual* savings, but *monthly* retirement income, adjust the number of periods accordingly.

In most published annuity formulas, the annual interest rate, compounded annually, is assumed. But since interest these days is usually compounded daily, the program uses effective interest rates. These are the rates that exist, assuming compounding is done more often than annually. The effective interest rate for daily compounding is supplied by the formula:

$$I_{eff} = \left[\frac{I_{nom}}{360} + 1 \right]^{360} - 1$$

Where I_{nom} is the stated annual interest rate.

For example, a 10 percent nominal rate provides an effective rate of 10.5156 percent when compounded daily. In some cases, such as some passbook accounts, interest may be compounded quarterly. In such cases, the two values of 360 in the formula are changed to 4. For monthly compounding, change each 360 to 12.

The power of compound interest is a source of amazement to people unfamiliar with its workings. As an example, suppose you are 30 years old and plan to establish your own IRA to which you will deposit $2,000 annually for 38 years. If you just put the money in a coffee can, at the end of 38 years you would have $76,000—a tidy sum. But at 10 percent compound interest, your savings will grow to $831,599 during the same 38 years. And that could give you a retirement of $8,313.53 per month, assuming 10 percent interest, for 20 years.

For another example, suppose you decide you will need to supplement your retirement by an additional $1,000 per month for 20 years from retirement age. Assuming you are 30 years old, and assuming you will retire at 68, and further assuming you will earn 10 percent interest both on savings and retirement annuity, you would need to save only $240.57 per year. With annual savings that low, you probably could not earn 10 percent interest. Let's assume you can only earn 5 1/4 percent in a passbook account. Plugging that into the program, you find you will need to save $850.00 per year.

Of course, smart savers will watch their savings closely, transferring funds

from passbook accounts to savings certificates, or to high yield money market funds as the principal increases.

The program lets you experiment with variables as often as you like. You may choose between having a known monthly retirement or a known annual savings. Once you make this choice, you are asked five questions. When your answers are complete, the program computes and displays the desired data. By entering 1 at the end of the computation, you can repeat the calculation using different variables. If you want to change only one variable, press ENTER in response to questions whose answers are to remain the same. By entering 2, you return to the main menu. This lets you choose between program functions. Entering a 3 ends the program run.

You can test the accuracy of the formulas you have typed by running the program both ways, using the same data. For example, suppose you know you want to have a monthly supplementary retirement income of $200. You select 1 from the menu. Let's assume you want your retirement annuity to last 15 years, earning 12 percent interest, and that you have 25 years to go until retirement, during which time your savings will earn 8 percent. Entering these variables, you find that you will need to save $208.96 per year, earning a total principal of $16,015.30. To see if you have entered the formulas accurately in your program first return to the main menu (option 2), then select option 2 ("What monthly retirement can I expect?"). Then, in answer to the first question, enter $208.96, the amount you calculated when you chose 1 from the menu. Plug in the same variables as you entered before for times and interest rates. You should come up with $200.00 per month annuity and $16,015.30 principal. Because of rounding, you may not come up with these figures to the penny, but, if you entered the formulas carefully paying close attention to the parentheses, you will be quite close.

The program is particularly valuable to persons with IRA and Keogh plans who anticipate a fixed annual savings. It shows the power of compound interest and the advantage of starting your savings program early in life.

Program Listing. *Retirement program*

Encyclopedia Loader

```
10 'RETIREMENT PROGRAM
20 'BY R. L. CONHAIM
30 E$ = "**$##,###,###.##"
40 ON ERROR GOTO650
50 CLS:PRINT@192,"** PLANNING YOUR RETIREMENT **"
60 PRINT:PRINT"1. WHAT WILL I NEED TO SAVE EACH YEAR?
   (IF YOU HAVE DETERMINED MONTHLY RETIREMENT)"
70 PRINT:PRINT"2. WHAT MONTHLY RETIREMENT CAN I EXPECT?
   (IF YOU HAVE DETERMINED ANNUAL SAVINGS)"
80 PRINT:PRINT:PRINT:INPUT "ENTER A CHOICE (1 OR 2)";D1
90 IF D1 = 1 GOTO110
100 IF D1 = 2 GOTO330ELSEGOTO50
110 CLS:PRINT@256,;
120 INPUT"HOW MUCH MONTHLY RETIREMENT INCOME DO YOU WANT";A1
130 IF A1=0 GOSUB590: GOTO120
140 INPUT "FOR HOW MANY YEARS AFTER RETIREMENT";N1
150 IF N1=0 GOSUB590: GOTO140
160 INPUT "AT WHAT ANNUAL INTEREST RATE (WHOLE NUMBER)";I3
170 IF I3=0 GOSUB590: GOTO160
180 INPUT "HOW MANY YEARS TO RETIREMENT";N2
190 IF N2=0 GOSUB590: GOTO180
200 INPUT "WHAT ANNUAL INTEREST RATE DO YOU EXPECT ON YOUR SAVINGS
   (WHOLE NUMBER)";I1
210 IF I1=0 GOSUB590: GOTO200
220 GOSUB550
230 P1 =((((1 + I4)[(N1 * 12))-1)/(((1 + I4)[(N1 * 12)) * I4)) * A1
240 S1 = (P1 *I2)/(((1 + I2)[N2)-1)
250 CLS:PRINT@256,"YOU WILL NEED TO SAVE ";USING E$;S1:PRINT"PER YEAR
     FOR";N2;"YEARS.":PRINT:PRINT"YOUR TOTAL SAVINGS WITH INTEREST WIL
     L BE ";USING E$;P1
260 D2=0 : PRINT@640,"1. DO ANOTHER CALCULATION"
270 PRINT"2. RETURN TO MAIN MENU"
280 PRINT"3. END PROGRAM RUN"
290 PRINT:INPUT"ENTER A CHOICE (1, 2, OR 3)"; D2
300 IF D2=0 OR D2>3 GOTO260
310 ON D2 GOTO110,50,630
320 '
330 CLS:PRINT@256,;
340 INPUT"HOW MUCH WILL YOU SAVE EACH YEAR (NO COMMAS)"; S1
350 IF S1=0 GOSUB590: GOTO340
360 INPUT "AT WHAT ANNUAL INTEREST RATE (WHOLE NUMBER)";I1
370 IF I1=0 GOSUB590: GOTO360
380 INPUT "HOW MANY YEARS TO RETIREMENT";N2
390 IF N2=0 GOSUB590: GOTO380
400 INPUT "FOR HOW MANY YEARS AFTER RETIREMENT ARE BENEFITS TO LAST";N
     1
410 IF N1=0 GOSUB590: GOTO400
420 INPUT "WHAT ANNUAL INTEREST RATE DO YOU EXPECT DURING RETIREMENT
   (WHOLE NUMBER)";I3
430 IF I3=0 GOSUB590: GOTO420
440 GOSUB550
450 P1 = ((((1 + I2)[N2)-1)/I2) * S1
460 A1 = ((P1 * I4) * (1 + I4)[(N1 * 12))/(((1 + I4)[(N1 * 12)) -1)
470 CLS:PRINT@256,"AT RETIREMENT YOU WILL RECEIVE ";USING E$;A1:PRINT
     "PER MONTH FOR";N1;"YEARS.":PRINT:PRINT "YOUR TOTAL SAVINGS WILL
     BE ";USING E$;P1
480 D3=0 : PRINT@640,"1. DO ANOTHER CALCULATION"
490 PRINT"2. RETURN TO MAIN MENU"
500 PRINT"3. END PROGRAM RUN"
510 PRINT:INPUT"ENTER A CHOICE (1, 2, OR 3)"; D3
520 IF D3=0 OR D3>3 THEN480
530 ON D3 GOTO330,50,630
540 END
550 I2 = (((I1/36000)+1)[360)-1
560 I4 = ((((I3/36000)+1)[360)-1)/12
570 RETURN
580 '
```

Program continued

```
590 REM        ERROR MESSAGE SUBROUTINE
600 PRINT"** TRY AGAIN! VALUE MUST BE GREATER THAN ZERO! **"
610 RETURN
620 '
630 CLEAR 50:CLS:PRINT"RUN ENDED":ON ERROR GOTO 0:PRINT : END
640 '
650 REM        ERROR TRAP
660 IF ERR/2+1=6 PRINT:PRINT"**** CAN'T PROCESS -- VALUES ENTERED ARE
    TOO HIGH ****
PRESS <ENTER> TO RE-START PROGRAM":INPUT XX:RESUME10
670 RESUME10
```

INTERFACE

Atari Joystick to TRS-80 Interface

Atari Joystick to TRS-80 Interface

by Carl Van Wormer

This article will show you how to build an interface to connect the Atari joystick to your Model I TRS-80. It also includes a short checkout program (Program Listing 1) and a BASIC program that uses an Atari joystick to simulate the doodling of Skedoodle or Etch-a-Sketch (Program Listing 2).

A joystick provides an alternative means of input for people who cannot use the keyboard, including pre-alphabet children and the handicapped. It gives a more responsive interface for certain game programs that makes them easier to operate. There are several excellent arcade games available that work very well with this joystick interface. It also keeps excitable gamesters from beating brutally on your keyboard when the enemy gets close.

Background

This interface attaches to the expansion port on either the CPU or the expansion interface (on the expansion interface, the expansion port is called the bus extension or screen printer port). It is a minimum hardware interface and has the advantage of being very simple and inexpensive. The switches inside the joystick are connected to the IN* (port input, low true) line and, through data lines, on the expansion port connector. When the BASIC INP(0) function or any of the machine-code port input instructions are used, the IN* line goes low, and the data bus is read by the CPU. Normally the data lines are at a logic high state when no device is addressed on the bus, but if switch contact is made in the joystick, the logic low on the IN* line is connected (through the diodes) to the data bus and is read as valid data input (see Figure 1).

This simple interface has two possible disadvantages. You may encounter difficulties if other port-mapped inputs on this bus are addressed while the joystick is used. This means you shouldn't use the joystick while you are trying to load a cassette program or use the RS-232 port, unless you are experimenting with arbitrary program/data manipulation. In most cases you will not encounter this problem, since the joystick is mainly used to play games that make no other use of the input ports. The joystick interface does not work with some non-TRS-80 expansion interfaces. If you have any non-TRS-80 equipment connected to your system and want to see if you will be able to use this interface, try this test. Turn on your computer, and while in

BASIC, type PRINT INP(0) and press ENTER. If the screen shows 255, everything should work. If the screen shows some other value, it indicates that the data bus has been terminated for better noise immunity. Unfortunately, this also means that my low budget interface won't work. To interface to a properly terminated data bus requires tri-state buffers and decode logic along with a power supply, which is beyond the scope of this project.

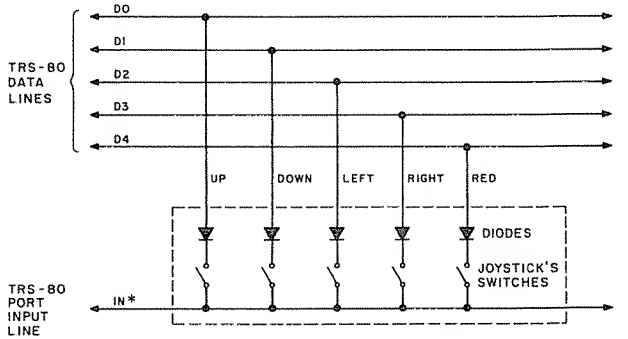


Figure 1. *Schematic diagram showing the Atari joystick interface*

Construction Methods

The goal of this project is to make the proper connections between the joystick and your computer. Three ways of doing this will be presented. For those of you who haven't violated your warranty by opening up the keyboard box, I suggest either of the first two methods. The first method consists of making a junction box for terminating the cable and mounting the DB-9 connector and diodes. The second method places the diodes inside a standard connector shell. The third approach is to mount the DB-9 connector on the TRS-80 case and connect the diodes directly to the data and control lines on the TRS-80 printed circuit board.

Construction Method 1

All parts except the DB-9 connector, the cable clamp, and the Atari joystick can be bought at Radio Shack for $18 to $24 (see Table 1). Make a cable clamp from a small piece of scrap aluminum or steel with two holes 5.5 centimeters (2 1/4") apart. Prepare the box lid by drilling and filing holes to mount the DB-9 connector and the cable clamp as shown in Figure 2. Mount the DB-9 connector in the box lid with #4 screws and nuts.

Press the 40-pin edge connector onto the ribbon cable, after being sure of proper alignment, by gently crushing the two halves between two blocks in a vise as shown in Figure 3. Prepare the wires as shown in Figure 4 and clamp the cable to the box as shown in Figure 5. Solder the diodes between the cable and the DB-9 connector as shown in Figure 6, then slide the heat

shrink tubing over the diode-wire joint and heat shrink it. After filing a notch in the top side of the plastic box for the cable entry, put the lid on the box, connect the 40-pin edge connector to the expansion port, and plug in an Atari joystick. Run the checkout program to verify proper operation (see Program Listing 1).

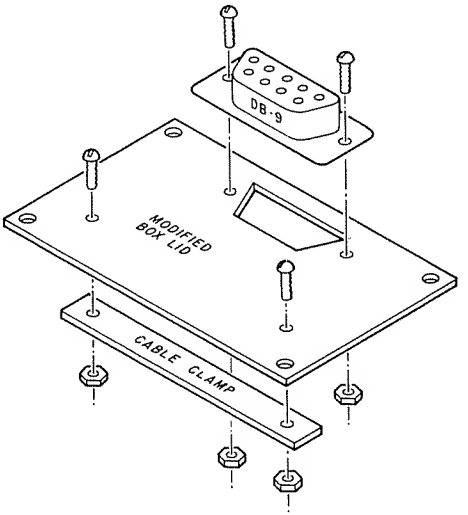


Figure 2. *Mounting DB-9 connector and cable clamp to box lid*

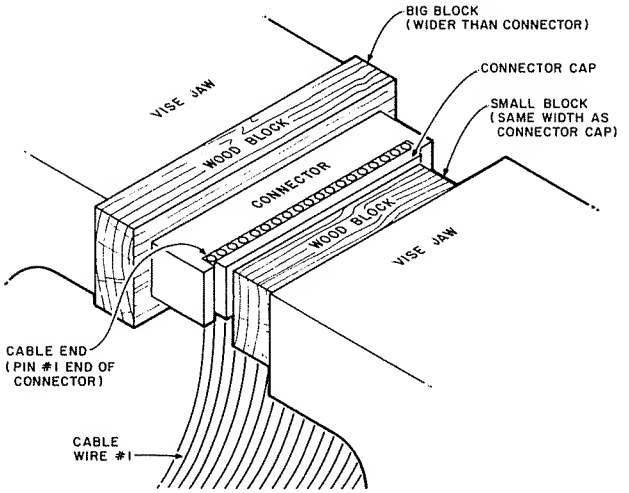


Figure 3. *Carefully crushing connector cap onto cable*

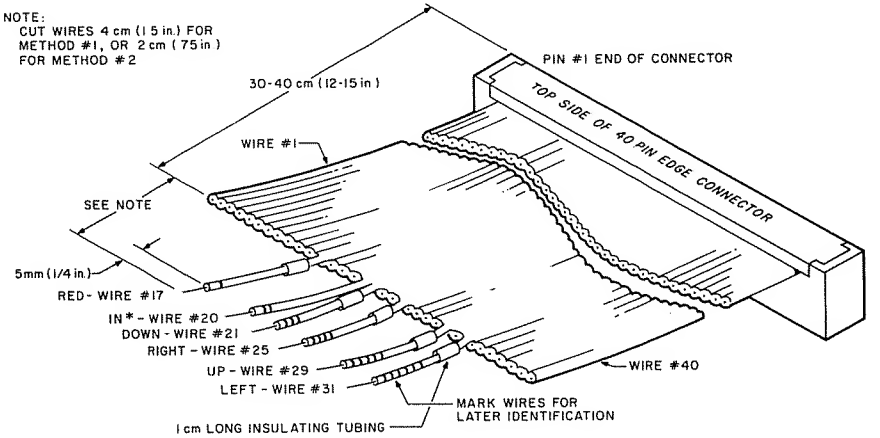


Figure 4. *Cable preparation*

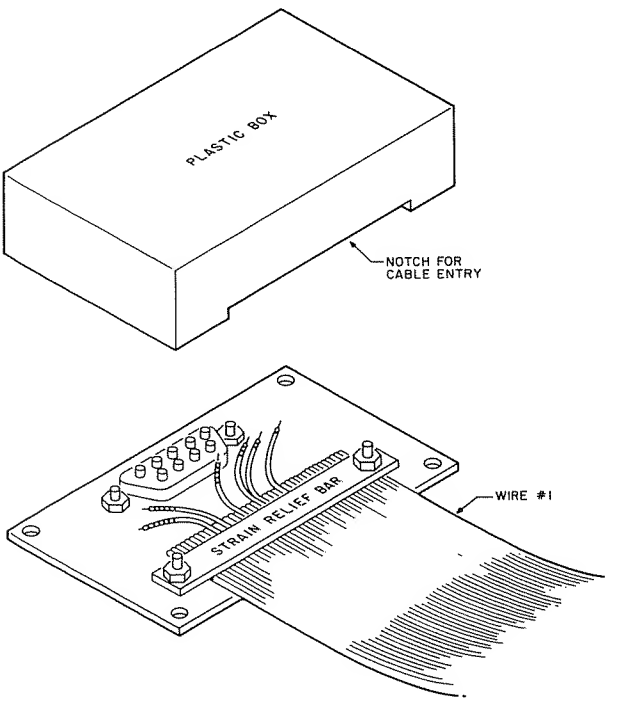


Figure 5. *Connecting cable to box lid with strain relief bar*

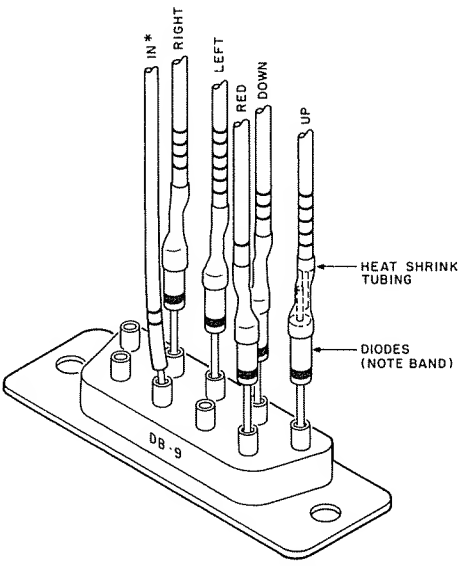


Figure 6. *Connecting DB-9, diodes, and wires*

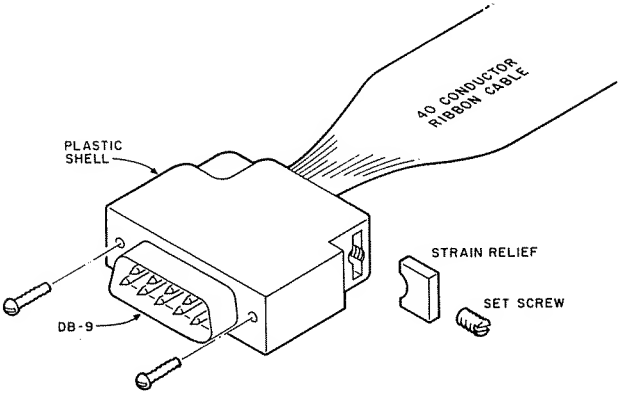


Figure 7. *Attaching the DB-9 connector with shell to the ribbon cable*

Construction Method 2

A kit containing the parts used in construction method 2, except the Atari joystick, is available for $16 (see Table 2). This method is similar to the previous method but requires less work. Instead of building a box to hold the connector and diodes, you connect the DB-9 connector directly to the ribbon cable. The cable must be prepared as in Figure 4, then folded in half three times to make a round bundle small enough to fit through the DB-9 shell (Figure 7). It is important to mark the cable as shown in Figure 4 since it is

very hard to determine which wire goes where in this tight bundle. After the cable is carefully pushed through the shell, solder the diodes to the DB-9 connector and the ribbon cable as shown in Figure 6, then slide the heat shrink tubing over the diode-wire joint and heat shrink it. Carefully pull back on the cable while pushing the DB-9 into place on the shell, then insert the two screws and tighten them until the connector is secured to the shell. Plug the Atari joystick into the DB-9 and connect the 40-pin edge connector to the expansion port on the computer and verify proper operation (see Program Listing 1). If everything works properly, insert the plastic strain relief and tighten the set-screw until the cable is secured.

Quantity	Part	Comments
1 foot	40-conductor ribbon cable	R/S #278-771 (5 ft)
1	40-pin connector	R/S #276-1558
5	Schottky diodes or	R/S #276-1124 (2/pack)
	Germanium diodes	R/S #276-1123 (10/pack)
1	Box	R/S #270-230
1	Heat shrink tubing	R/S #278-1627 (package)
4	#4/40 screws	R/S #64-3011 (box)
4	#4/40 nuts	R/S #64-3018 (box)
1	DB-9 male connector	

Total cost of Radio Shack parts—$18 to $24 (with parts left over)
Cost of DB-9 connector—$2 at an electronics parts store
Atari joystick—$10 at any Atari repair center or most Atari retail outlets

Table 1. *List of components needed when using construction method 1*

Construction Method 3

For those of you who don't want wires and boxes around your computer, this method is more appropriate and cheaper. You simply attach the DB-9 connector to the side of your expansion interface or your CPU box. The diodes then go from the DB-9 connector to the signal lines on the printed circuit board (see your *Hardware Technical Manual*, Radio Shack #26-2103). Make holes in your plastic box (either the keyboard or the expansion interface) of the correct size to mount the DB-9 connector. Carefully connect wires from the diodes to the signal lines as shown in Figures 1 and 6. Reassemble the box and verify operation using Program Listing 1.

Troubleshooting

If you connect this device to the computer and the computer goes crazy every time you move the joystick or press the red button, you probably have one or more diodes in backward. This causes the data lines to go to logic high whenever the INP(0) function is not used and a switch contact is made. The

cure is to put all the diodes in the right way—check Figure 6 for diode banded end direction.

Quantity	Part
1 foot	40-conductor ribbon cable
1	40-pin connector
5	Schottky diodes or Germanium diodes
1	DB-9 male connector
1	DB-9 plastic shell

Note: A complete kit of the above parts is available from Norvac Electronics, 12905 SW Beaverdam Rd., Beaverton, OR 97005. The price is $15.95, which includes postage for shipping in the continental United States. This package is for construction method 2 and has the 40-pin edge connector already connected to the cable. It does not include the Atari joystick.

Table 2. *List of components needed for construction method 2*

If one direction of the joystick does not work, this indicates an open circuit in that line. You should trace that particular line from one end to the other, looking for the bad connection. If two control actions have the same effect, this is most likely caused by a short between the two lines. Check for solder blobs at the connections to the DB-9. If you made your own connection between the cable and the 40-pin edge connector, and you can't find any obvious errors at the other end, the cable may be crooked or the connector may not be crimped onto the cable properly. A careful examination should find the problem and indicate the proper solution. If it still doesn't work, use an ohmmeter to help locate the problem. One final possibility is that the Atari joystick may be defective. This happens most often on old ones that have seen many hours of combat.

Program Listing 1. *Atari joystick checkout program*

```
1001 REM  ************************************************
1101 REM  ************** LISTING # 1 *********************
1201 REM  ************** JOYTEST/BAS *********************
1301 REM  ************************************************
1401 REM   ATARI JOYSTICK CHECKOUT PROGRAM
       CARL B. VAN WORMER
       359 NE HILLWOOD DR
       HILLSBORO,OR 97123
       01/15/82
1501 DEFINT A-Z
1601 A = INP(0) : REM  READ PORT DATA
1701 B = NOT A : REM  INVERT BITS FOR POSITIVE LOGIC (HI TRUE)
1801 C = B AND 31 : REM  MASK OFF UNWANTED BITS
1901 CLS
2001 PRINT@0, "JOYSTICK CONTACT TEST:";
2101 IF C AND 1 THEN PRINT@351, "UP";
2201 IF C AND 2 THEN PRINT@862, "DOWN";
2301 IF C AND 4 THEN PRINT@593, "LEFT";
2401 IF C AND 8 THEN PRINT@619, "RIGHT";
2501 IF C AND 16 THEN PRINT@337, "RED";
2601 GOTO1601
```

Program Listing 2. *Using Atari joystick to simulate doodling of Skedoodle or Etch-a-Sketch*

```
10 REM  ************************************************
20 REM  ************** LISTING # 2 *********************
30 REM  **************  ATARI/BAS  *********************
40 REM  ************************************************
50 REM  ROUTINE TO ACCEPT INPUT FROM ATARI JOYSTICK (PORT D0-D4)
      AND FROM KEYBOARD ARROWS AND SPACE BAR THEN DRAW WITH IT
       BY CARL B. VAN WORMER
       359 NE HILLWOOD DR.
       HILLSBORO, OR 97123
       01/04/82

60 REM  DATA WORD READ IN FROM INP(0) IS SEEN AS LOW TRUE (0)
       FOR BIT POSITION OF THAT SWITCH
70 REM  DATA WORD READ IN FROM KEYBOARD 'MEMORY' IS SEEN AS
       HI TRUE (1) FOR BIT POSITION OF THAT KEY.  KEYBOARD
       IS ADJUSTED TO MATCH JOYSTICK WITH DIVISION BY 8
80 REM  BIT POSITIONS IN DATA WORD ARE AS FOLLOWS:
      D0=UP, D1=DOWN, D2=LEFT, D3=RIGHT, D4=RED BUTTON (SPACE)

90 DEFINT A-Z
100 CLS : PRINT CHR$(23);"  USE ARROW KEYS AND SPACE BAR" :
    PRINT@204," OR ATARI JOYSTICK" :
    PRINT@963,"SPACE AND CLEAR CLEARS SCREEN";CHR$(28)
110 PRT = 31 AND NOT INP(0) : REM  REMOVE TOP BITS OF INVERTED
              PORT INPUT TO MAKE IT LOOK LIKE KEYBOARD INPUT
120 KEYBD = PEEK(14400) : REM  ARROWS AND SPACE BAR ROW
130 IF POINT (X,Y) THEN RESET (X,Y) ELSE SET (X,Y) : REM INVERT
140 FOR Q=1 TO 5 : NEXT Q : REM  DOT FLASH BRITENESS TIME DELAY
150 IF POINT (X,Y) THEN RESET (X,Y) ELSE SET (X,Y) : REM INVERT
160 IF PRT=0 AND KEYBD<8 THEN GOTO 110 : REM  NO PUSH-TRY AGAIN
170 IF KEYBD=130 THEN CLS : KEYBD = 0 :
                        REM  IF SPACE AND CLEAR ARE PRESSED
180 KEYBD = KEYBD/8 : REM  MOVE D3-7 TO D0-4 TO MATCH JOYSTICK
190 BITS = PRT OR KEYBD : REM  EITHER JOYSTICK OR KEYBOARD
200 IF BITS >= 16 THEN PSHBTN=1 : BITS = BITS-16
210 ON BITS GOSUB 1010,1020,1090,1030,1050,1070,1030,1040,1060,
              1080,1040,1090,1010,1020,1090
220 IF X<0 THEN X = 0 ELSE IF X>127 THEN X = 127
230 IF Y<0 THEN Y = 0 ELSE IF Y>47 THEN Y = 47 :
                                  REM  CHECK FOR X,Y LIMITS
```

```
240 IF PSHBTN=1 THEN SET (X,Y)
250 IF PSHBTN=0 THEN RESET (X,Y)
260 PSHBTN = 0
270 GOTO 110 : REM  GO BACK AND DO IT AGAIN
1000 REM  ************************* TURN DATA INTO MOTION ***
1010 Y = Y-1 : RETURN : REM  UP
1020 Y = Y+1 : RETURN : REM  DOWN
1030 X = X-1 : RETURN : REM  LEFT
1040 X = X+1 : RETURN : REM  RIGHT
1050 GOSUB 1010 : GOSUB 1030 : RETURN :REM  UP-LEFT
1060 GOSUB 1010 : GOSUB 1040 : RETURN : REM  UP-RIGHT
1070 GOSUB 1020 : GOSUB 1030 : RETURN : REM  DOWN-LEFT
1080 GOSUB 1020 : GOSUB 1040 : RETURN : REM  DOWN-RIGHT
1090 RETURN : REM  CONTRADICTION...UP+DOWN OR RIGHT+LEFT
1100 REM  ********************** END OF SUBROUTINE ******
```

TUTORIAL

TRS-80 Cryptographer
Lazy Logic Trainer
Screen Status Byte

TRS-80—Cryptographer

by Allan S. Joffe W3KBM

The problem in getting the TRS-80 to produce cryptograms is to take a message in English and turn it into a form that is encoded so that it must be decoded to regain the original intelligence.

Consider the following example.

```
 1 REM  UNIVERSAL ENCODE AND DECODE PROGRAM
 3 CLEAR 300
 5 CLS
10 INPUT "MESSAGE TO ENCODE OR DECODE";A$
15 CLS
20 PRINT A$
30 FOR X = 15360 TO 15680
35  REM  LINE 30 GIVES SWEEP OF 5 LINES OF SCREEN SO ENTIREMESSAGE
    IS SCANNED.  IT AVOIDS NEED TO FILL BLANKS TO ENDOF LINE, WITHO
    UT  WHICH YOU CAN MISS CHARACTERS.
40  POKE X,91 - PEEK(X)
50  IF PEEK(X) = 59
     THEN
      POKE X,32:
      GOTO 60
60  NEXT X
70 END
```

Line 10 asks for a message, which is then assigned to string variable A$. You must remember to enclose the message in quotation marks if it contains any commas or colons.

Line 30 could also be written as:

```
30  FOR A = 15360 TO 15360 + LEN(A$)
```

However, this would cause the program to fail if you used the down-arrow key to obtain a line feed in your message. This is further described in the REMark in line 35.

Line 40 POKEs the encoded character back into video memory. Line 50 deletes any unwanted semicolons that would otherwise appear in the display.

After typing in the program, type RUN and press ENTER. You are prompted to enter a message. The program then encodes the message and displays it at the top of the screen. For example, if you type in THE QUICK BROWN FOX, the program returns the message as: GSV JFRXP YILDM ULC. If you run the program again and type in the scrambled or encoded message, it returns: THE QUICK BROWN FOX.

This method of encryption is not very sophisticated. You can observe this by running the program and typing in the alphabet. The encoded message appears as the alphabet reversed: A = Z, B = Y, C = X, and so forth. Manually decoding such a message is easy to do by applying letter frequency methods. "Letter frequency of occurrence" simply means how often a given letter appears in a block of text. The letter frequency of occurrence in the English language follows this general sequence:

E T R I N O A S D L C H F U P M Y G W V B X K Q J Z

By studying the encoded message, character by character, you can decode it by determining how often each letter appears.

The following program overcomes the deficiencies of the first program.

```
  5 REM  LISTING # TWO
  7 CLS
  8 CLEAR 255
 10 INPUT "SECRET MESSAGE";A$
 20 FOR X = 1 TO LEN(A$)
 30  B$ = MID$(A$,X,1)
 40  IF ( ASC(B$) > 35) - (X < LEN(A$)) PRINT B$;:
      ELSE
       PRINT CHR$( ASC(B$) - INT( SQR(X / .59)));
 50  NEXT
 55 END
 95 REM  THIS IS THE DECODING SECTION
100 INPUT "TYPE IN SCRAMBLED MESSAGE";A$
105 CLS
110 FOR X = 1 TO LEN(A$)
120  B$ = MID$(A$,X,1)
130  IF ( ASC(B$) > 35) - (X < LEN(A$)) PRINT B$;:
      ELSE
       PRINT CHR$( ASC(B$) + INT( SQR(X / .59)));
140  NEXT X
```

Lines 7–50 encode, and lines 100–140 decode. The encoding line 40 differs from line 130 in that the final arithmetic expression is subtracted to encode and added back to decode. It is the calculated value of this expression INT(SQR(X/.59)) that determines the character set of the scrambled message. With the values shown in the program, some of the alphanumeric data in the message to be scrambled will be turned into characters other than alphanumerics, for example #,*,? and so forth. This makes it harder to decode the message manually.

Before examining how this listing treats a message such as "Now is the time for all good men to come to the aid of their party," study line 40. Notice the minus sign between the first and second parenthetic expressions. This is to emphasize that the minus sign under discussion does not indicate the subtraction of one expression from another, but is actually the proper use of an undocumented capability of the TRS-80.

Below is an example of the typical scrambling of the test phrase.

```
NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID
MNU GP QEA PEI@ AJM <FF AII> F>G MH <GE= LG L@= 8@D
```

Immediately below the test phrase is the scrambled text produced by the en-

coding line of Program Listing 2. To run the decode section of the program, type RUN 100 (ENTER), and type this scrambled text into the computer. The message will be decoded and displayed.

Notice that in addition to introducing confusing symbols, it also eliminates the possibility of solving the phrase by the use of letter frequency counting. As an example, the letter O appears in the original text in the words now, good, come, and to. If you examine the encoded text you will see that the letter O in these positions has taken the value of N,J,I,H, and G, making letter frequency counting impossible.

The success of Program Listing 3 depends on the unique quality of the TRS-80 graphics symbols. This program converts the alphanumeric information of a secret message into graphics characters.

```
1 REM ENCODING,DECODING PROGRAM BY A.S.JOFFE  W3KBM
2 REM DECODING PORTION OF PGM STARTS AT LINE 180
3 REM MESSAGE ENCODED INTO GRAPHICS CHARACTERS
4 REM THIS IS LISTING # THREE
5 CLS
10 CLEAR 1000
15 DIM R(255)
20 INPUT "MESSAGE";A$
25 CLS
30 PRINT A$
35 PRINT
40 FOR X= 15360  TO 15360+LEN(A$)
50 POKE X,218-PEEK(X)
70 NEXT X
90 FOR J= 15360 TO X
95 L=L+1
100 R= PEEK(J)
110 R(L)=R
120 NEXT
130 FOR D= 1 TO L
140 PRINT CHR$(R(D));
145 B$=B$+CHR$(R(D))
150 NEXT
155 REM PUTTING DATA ONTO TAPE
160 PRINT#-1,L
170 PRINT#-1,B$
172 END
175 REM START OF DECODING PROGRAM FROM TAPE INPUT
180 CLS:INPUT"REWIND CASSETTE WITH MESSAGE ON IT.
PRESS <ENTER> WHEN READY"; X : CLS
200 INPUT#-1,L
210 INPUT #-1,B$
220 PRINT B$
230 FOR X= 15360 TO 15360+L
250 POKE X,218-PEEK(X)
270 NEXT X
```

The key to your success in transmitting secret messages using all graphics characters is shown in Program Listing 3. Lines 5–120 change the text string or message into graphics characters. To minimize tape loading time, 130–150 pack the data into an array and form B$. This puts all of the text—now converted to graphics—into B$. Lines 160 and 170 do the actual work of putting the data onto your tape. When the tape stops, your encoded message is on tape. Type RUN 180 (ENTER), and the program prompts you to rewind the tape. Rewind your tape and press ENTER. The computer

reads the encoded tape, then decodes the message. You then see your original message come to life on the screen as the computer converts the graphics characters back into English.

Lazy Logic Trainer

by Archie P. Kelley Jr.

The purchase of my Model I TRS-80 years ago caused me to want to learn what goes on inside those gray plastic boxes. After taking a course in microcomputer design and experimenting with assembling and testing logic circuits, I created the Lazy Logic Trainer program (see Program Listing) to do with software what I should have been doing with the hardware on my breadboard.

Logic Model Creation

The first step in the implementation of the software model of the logic circuit is to prepare the logic circuit diagram and add appropriate identifiers to the logic gates and signals compatible with computer input. To illustrate this process, I've selected as an example the three-bit equality and relative magnitude detector circuit shown in Figure 1.

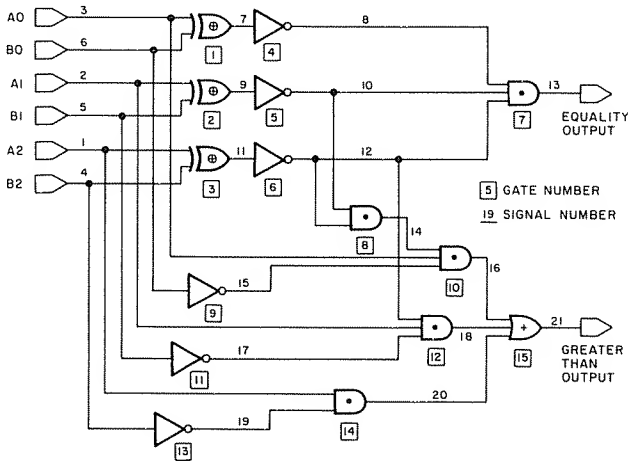


Figure 1. *Equality and relative magnitude detector logic diagram*

The circuit involves two major parts. The upper part of the circuit, which includes three EXCLUSIVE OR gates, three inverters, and an AND gate, compares the two three-bit binary numbers (A0–A2 and B0–B2 in Figure 1) for equality. A logical 1 output from the topmost AND gate signifies that the two three-bit input numbers are equal. The lower part of the circuit, which

includes three more inverters, four AND gates, and one OR gate, checks if the three-bit number A is greater than B. If so, a logical 1 is output from the bottom OR gate.

To prepare the circuit diagram for input, each of the 15 logic gates and 21 input and output signals must be numbered in succession, progressing from the input conditions to output conditions. An output from a gate cannot be developed until all of its input signals have been developed. One such numbering scheme is shown in Figure 1, although others are possible. Signal numbers 1 through 6 are reserved for the independent inputs to the circuit which represent the two three-bit numbers. Signal numbers 13 and 21 represent the desired outputs, giving the equality and greater-than conditions detected by the circuit.

The next step is to input the circuit model for computer simulation. After entering the total number of gates and signals (15 and 21 respectively), the program prompts you in sequence for the gate type, input signal numbers, and output signal numbers as shown in Figure 2.

```
MODEL INPUT—GATE NUMBER 1

AVAILABLE GATES:
1 -AND    2 -OR    3 -NOR    4 -NAND
5 -XOR    6 -INV   7 -GND    8 -J-KFF
9 -FADD
WHICH ONE (ENTER GATE NUMBER)? 5
INPUT SIGNAL NUMBERS (ENTER 0 IF NO SIGNAL):
                    INPUT 1 ? 3
                    INPUT 2 ? 6
OUTPUT SIGNAL NUMBERS (ENTER 0 IF NO SIGNAL):
                    OUTPUT 1 ? 7_
```

Figure 2. *Screen display during model input*

Once you enter the signal numbers and gate types for all gates, the program performs an error check of the model, which results in the screen output shown in Figure 3. The error check displays the number of gates and signals input and identifies independent input and output signals, that is, those signals which are not both an input and output to any gate. A search is then made for any unused signals, input or output signal numbers beyond the input range, outputs developed before inputs, or outputs from different gates which have the same signal number. Any of these errors indicates an improper model and prevents execution of the simulation, sending you to an input correction routine. In the example shown in Figure 3, the signals at gate 6 were entered incorrectly, causing an output to be developed before input and creating redundant output signal numbers.

```
                    MODEL ERROR CHECK
NUMBER OF GATES:  15
NUMBER OF SIGNALS:  21
INDEPENDENT OUTPUT NUMBERS:  13  21
INDEPENDENT INPUT NUMBERS:  1  2  3  4  5  6  12
UNUSED SIGNAL NUMBERS:
INPUT BEYOND SIGNAL RANGE:
OUTPUT BEYOND SIGNAL RANGE:
GATES AT WHICH OUTPUT DEVELOPED BEFORE INPUT:  6
OUTPUTS WHICH ARE DEVELOPED TWICE:  11
                  **FATAL ERROR DISCOVERED**
PRESS ENTER WHEN READY TO MAKE CORRECTIONS?__
```

Figure 3. *Screen display after error check*

Figure 4 displays the screen output provided by the model correction and review routine for the first 10 gates. Compare this with the schematic in Figure 1. For each logic gate, the logic type is displayed, along with input and output signal numbers. As detected by the error-trapping routine, the output signal number to gate 6 has been mislabeled upon entry and must be changed from 11 to 12. After the last page of the model review display, the program asks for any gate corrections. In this case, you must correct gate 6. The model is then checked again for errors. If there are none, the model goes to the input/output simulation routine.

Logic Model Output

The final product of all these labors is shown in Figure 5. The program asks you to enter each of the input signal states (either a logical 0 or 1) and

```
                  MODEL CORRECTION/REVIEW
GATE      TYPE      INPUTS          OUTPUTS
1         XOR       3  6            7
2         XOR       2  5            9
3         XOR       1  4            11
4         INV       7               8
5         INV       9               10
6         INV       11              11
7         AND       8  10  12       13
8         AND       10  12          14
9         INV       6               15
10        AND       3  14  15       16
PRESS ENTER FOR NEXT PAGE?__
```

Figure 4. *Screen display during model correction*

then simulates the circuit to generate the logical output conditions. In the example in Figure 5, signals 1 through 3 represent bits 2 to 0 of binary input A, and signals 4 through 6 represent bits 2 to 0 of binary input B. In the first row, binary 000 (decimal 0) has been entered for both A and B, resulting in an output of 1 for signal 13 (indicating equality of A and B) and 0 for signal 21. In the fourth row, binary 011 (decimal 3) for A and 010 (decimal 2) for B have been entered. The circuit simulation responds correctly with a logical 1 as signal 21, indicating that A has been decoded as greater than B. The circuit also responds correctly to other input conditions.

```
                    INPUT//OUTPUT
 1  2  3  4  5  6  //     13  21
 0  0  0  0  0  0  =>     1   0
 0  0  1  0  0  0  =>     0   1
 0  0  1  0  1  0  =>     0   0
 0  1  1  0  1  0  =>     0   1
 0  1  1  0  1  1  =>     1   0
 1  0  0  0  1  1  =>     0   1
 1  0  0  1  1  1  =>     0   0
 1  1  1  1  1  1  =>     1   0
 >

TYPE 0 OR 1 FOR EACH INPUT SIGNAL (HIT X TO CHANGE MODEL)
```

Figure 5. *Screen display during circuit simulation*

Program Description

The listing for the Lazy Logic Trainer program is broken into four major parts. First, a listing of key variables is provided. This is followed by the main program and two subprograms. Last are the logic module subprograms, one for each logic gate type.

Line 120 of the main program asks for the total number of gates and signals. This information is used to set up the major variable dimensions in line 130, followed by reading of the names of the available gate types and number of allowable inputs and outputs. The data for these read statements is contained in the logic gate modules described later.

The program asks you to enter information for each gate via the FOR-NEXT loop in line 190. The input model is then checked for errors by the FOR-NEXT loop routines in lines 200 through 390. If there are errors, the variable FLAG is set equal to one, signifying that the model must be corrected before execution.

If errors are detected, or if you want to review the model, the program branches to lines 400 through 490, which send to the screen the model which

has been input. The program prompts you for any changes in line 480. If changes are made, FLAG is again set to 1, forcing a return to the error check routine starting at line 200.

Lines 500–680 generate the model simulation and input and output conditions. You are told what input signals must be provided and what output signals by default will be displayed. You can specify additional outputs if you wish. Lines 560–590 prepare the screen display for input and output. Lines 600–620 prompt and receive input signal values via the INKEY$ function. Once a row of all independent input signals has been input, lines 630–680 march through the model gate-by-gate, setting input conditions and calculating the logical output via the logic module subprograms. Once all signal values have been calculated, the output signals are displayed, and the program loops back to get the next row of input. Each input signal condition is negated in lines 620 and 670 to be compatible with the bit complement logic of Level II BASIC, which uses − 1 and 0 rather than + 1 and 0.

The subprogram at line 900 displays a screen header, using the information stored in variable A1$. The second subprogram starting at line 1500 asks for information on the gate number stored in I and is called from lines 190 and 490 as required.

The logic modules, starting at line 2000, form the basis for simulating the output from each logic gate, based upon given input. Each subprogram starts with a data statement which defines the gate name and number of possible inputs and outputs. Next, the subprogram provides the logical BASIC statements necessary to convert the input conditions to the desired output conditions, returning to the main program with the output states stored in the variable S(O(I,1)).

Logic Gate Additions

There are nine different types of logic gates in the Program Listing, and there are provisions in the main program to add an additional six. You can use more complex gates in addition to those used in the example described above. For example, gate type 8 simulates a J-K flip-flop. J and K are represented by the first two inputs, and the flip-flop is clocked by inputting a one as the third gate input. The first and second outputs to the gate simulate the Q and NOT Q outputs of the real IC chip. Other clocked devices may similarly be hooked up to the same input signal to simulate clocked circuits. Gate type 9 simulates a full adder, with the two bits to be added input as the first two signals, and the carry bit input as the third. The first output signal then represents the sum output; the second represents the carry.

Other more sophisticated gate types are left to your invention. The program currently limits you to no more than three input and three output signals per gate, although it could be modified to handle more.

Program Listing. *Lazy Logic Trainer*

Encyclopedia Loader™

```
10 '=======================================================
11 '                    LOGIC TRAINER
12 '=======================================================
13 '
20 '  PREPARED BY:    ARCHIE P. KELLEY, JR.
25 '                   10020 CONNELL ROAD
30 '                  SAN DIEGO, CA. 92131
31 '                     JANUARY 1982
35 '
40 '  VARIABLES:
45 '           I(I,J)   = JTH INPUT SIGNAL TO GATE I
50 '           O(I,J)   = JTH OUTPUT SIGNAL FROM GATE I
51 '           G(I)     = ITH GATE TYPE
52 '           S(I)     = ITH SIGNAL VALUE (0 OR 1)
53 '           A$(I)    = DESCRIPTOR FOR GATE I
54 '           NI(I)    = NUMBER OF INPUTS ALLOWED TO GATE I
55 '           NO(I)    = NUMBER OF OUTPUTS ALLOWED TO GATE I
60 '           II(I)    = ITH SIGNAL TO BE INPUT
61 '           OI(I)    = ITH SIGNAL TO BE OUTPUT
70 '           NG       = NUMBER OF GATES IN MODEL
71 '           NS       = NUMBER OF SIGNALS IN MODEL
72 '           IL       = NUMBER OF SIGNALS TO BE INPUT
73 '           OL       = NUMBER OF SIGNALS TO BE OUTPUT
75 '
97 '=======================================================
98 '                    MAIN PROGRAM
99 '=======================================================
100 CLEAR(500):DEFINTB-Y
109 '               --- SET MODEL SIZE ---
110 A1$="    - LOGIC TRAINER -":GOSUB900
120 INPUT"NUMBER OF GATES";NG:INPUT"NUMBER OF SIGNALS";NS
130 DIM I(NG,3),O(NG,3),G(NG),S(NS)
140 DIM A$(15),NI(15),NO(15),II(10),OI(5)
150 I=1
160 READ A$(I):IFA$(I)="END"THEN180
170 READ NI(I),NO(I):I=I+1:GOTO160
180 GN=I-1
189 '            --- INPUT MODEL BY GATE ---
190 FORI=1TONG:A1$="MODEL INPUT - GATE NUMBER "+STR$(I):GOSUB900:GOSUB
    1500:NEXTI
199 '            --- CHECK MODEL FOR ERRORS ---
200 A1$="MODEL ERROR CHECK":GOSUB900
210 PRINT"NUMBER OF GATES:    ";NG:PRINT"NUMBER OF SIGNALS: ";NS
220 OL=1:PRINT"INDEPENDENT OUTPUT NUMBERS:";:FORI=1TONS:FORJ=1TONG:FOR
    K=1TO3:IFI=I(J,K)THEN240
230 NEXT K,J:PRINT I;:OI(OL)=I:OL=OL+1
240 NEXTI:PRINT:OL=OL-1
250 IL=1:PRINT"INDEPENDENT INPUT NUMBERS: ";:FORI=1TONS:FORJ=1TONG:FOR
    K=1TO3:IFI=O(J,K)THEN270
260 NEXT K,J:PRINT I;:II(IL)=I:IL=IL+1
270 NEXTI:PRINT:PRINT:IL=IL-1
280 FLAG=0:PRINT"UNUSED SIGNAL NUMBERS: ";:FORI=1TONS:FORJ=1TONG:FORK=
    1TO3:IFI(J,K)=I OR O(J,K)=I THEN300
290 NEXTK,J:PRINT I;:FLAG=1
300 NEXTI:PRINT
302 PRINT"INPUT BEYOND SIGNAL RANGE: ";:FORI=1TONG:FORJ=1TO3:IFI(I,J)>
    NSTHENPRINT I(I,J);:FLAG=1
304 NEXTJ,I:PRINT
306 PRINT"OUTPUT BEYOND SIGNAL RANGE: ";:FORI=1TONG:FORJ=1TO3:IFO(I,J)
    >NSTHENPRINT O(I,J);:FLAG=1
308 NEXTJ,I:PRINT
310 PRINT"GATES AT WHICH OUTPUT DEVELOPED BEFORE INPUT: ";:FORI=1TONG:
    FORJ=1TO3:FORK=1TO3
320 IFO(I,J)=0 OR I(I,K)<O(I,J) THEN 340
330 PRINT I;:FLAG=1
340 NEXT K,J,I:PRINT
345 IFNG=1THEN380
350 PRINT"OUTPUTS WHICH ARE DEVELOPED TWICE: ";:FORI=1TONG-1:FORJ=1TO3
```

```
    :FORK=I+1TONG:FORL=1TO3:IFO(I,J)=O(K,L) AND O(I,J)>O THEN360 ELSE
     370
360 PRINT O(I,J);:FLAG=1
370 NEXT K,J,I:PRINT
380 IFFLAG>OTHENPRINT:PRINT,"** FATAL ERROR DISCOVERED **":INPUT"PRESS
     ENTER WHEN READY TO MAKE CORRECTIONS";A2$:GOTO400
390 PRINT:PRINT,"** NO ERRORS DISCOVERED **":INPUT"DO YOU WISH TO REVI
    EW MODEL OR MAKE CHANGES(YES/NO)";A2$:IFLEFT$(A2$,1)="N"THEN500
399 '              --- CORRECT OR REVIEW MODEL ---
400 A1$="MODEL CORRECTION/REVIEW":GOSUB900
410 PRINT"GATE","TYPE","INPUTS","OUTPUTS"
420 FORI=1TONG:PRINTI,A$(G(I)),:FORJ=1TO3:IFI(I,J)=OTHEN440
430 PRINT I(I,J);
440 NEXTJ:PRINT,:FORJ=1TO3:IFO(I,J)=OTHEN460
450 PRINT O(I,J);
460 NEXTJ:PRINT:IFINT(INT(I/10)*10)=ITHENINPUT"PRESS ENTER FOR NEXT PA
    GE";A2$
470 NEXTI
480 PRINT:INPUT"DO YOU WISH TO MAKE CORRECTIONS(YES/NO)";A2$:IFLEFT$(A
    2$,1)="Y"THEN490
485 IFFLAG=1THEN200 ELSE 500
490 FLAG=1:INPUT"WHICH GATE";I:A1$="CORRECTION - GATE "+STR$(I):GOSUB9
    00:GOSUB1500:GOTO400
499 '                  --- OUTPUT PREPARATION/DISPLAY ---
500 A1$="INPUT // OUTPUT":GOSUB900
510 PRINT"THE FOLLOWING INPUT SIGNALS WILL BE PRINTED:":FORI=1TOIL:PRI
    NT I1(I);:NEXTI:PRINT
540 PRINT:PRINT"THE FOLLOWING OUTPUT SIGNALS WILL BE PRINTED:":FORI=1T
    OOL:PRINT O1(I);:NEXTI:PRINT
550 INPUT"HOW MANY WOULD YOU LIKE TO ADD(MAX=5)";NA:IFNA=OTHEN570
560 FORI=1TONA:OL=OL+1:INPUT"SIGNAL NUMBER";O1(OL):NEXTI
570 GOSUB900:Z$="##    ":FORI=1TOIL:PRINT USING Z$;I1(I);:NEXTI
580 PRINT"//  ";:FORI=1TOOL:PRINT USING Z$;O1(I);:NEXTI:PRINT
590 PRINT@896,STRING$(63,"="):PRINT"TYPE 0 OR 1 FOR EACH INPUT SIGNAL(
    HIT X TO CHANGE MODEL)";:PRINT@192,"";
600 PRINT">";CHR$(24);:FORI=1TOIL
610 A$=INKEY$:IFA$=""THEN610
612 IFA$="X"THEN400
615 NN=VAL(A$):IFABS(NN)>1THEN610
620 S(I1(I))=NN*-1:PRINT USING Z$;NN;:NEXTI
630 PRINT"=>  ";
640 FORI=1TONG:I1=S(I(I,1)):I2=S(I(I,2)):I3=S(I(I,3))
650 ON G(I) GOSUB 2000,2100,2200,2300,2400,2500,2600,2700,2800,2900,30
    00,3100,3200,3300,3400
660 NEXTI
670 FORI=1TOOL:PRINT USING Z$;-S(O1(I));:NEXTI:PRINT
680 GOTO600
890 '
891 '=======================================================
892 '                       SUBPROGRAMS
893 '=======================================================
894 '
899 '          --- DISPLAY SCREEN HEADER IN A1$ ---
900 CLS:PRINT,A1$:PRINTSTRING$(63,"="):RETURN
1498 '
1499 '             --- INPUT MODEL BY GATE I ----
1500 PRINT"AVAILABLE GATES:":FORJ=1TOGN:PRINTJ;" - ";A$(J),:NEXTJ:PRIN
     T:PRINT
1510 INPUT"WHICH ONE(ENTER GATE NUMBER)";G(I):IFG(I)>GNPRINT"NO SUCH G
     ATE - TRY AGAIN":GOTO1510
1520 J=G(I):NI=NI(J):NO=NO(J)
1530 GOSUB900:PRINT"GATE TYPE: ";A$(J):PRINT
1540 IFNI=OTHEN1570
1550 PRINT"INPUT SIGNAL NUMBERS(ENTER 0 IF NO SIGNAL):"
1560 FORJ=1TONI:PRINT,"INPUT";J;:INPUTI(I,J):NEXTJ:PRINT
1570 PRINT"OUTPUT SIGNAL NUMBERS(ENTER 0 IF NO SIGNAL):"
1580 FORJ=1TONO:PRINT,"OUTPUT";J;:INPUTO(I,J):NEXTJ
1590 RETURN
1990 '=======================================================
1991 '                    LOGIC MODULES
```

Program continued

```
1992 '=====================================================
1998 '
1999 '                    --- AND GATE ---
2000 DATA AND,3,1
2010 IF I(I,3)=0 THEN 2030
2020 O1=I1 AND I2 AND I3:GOTO2040
2030 O1=I1 AND I2
2040 S(O(I,1))=O1:RETURN
2098 '
2099 '                    --- OR GATE ---
2100 DATA OR,3,1
2110 IF I(I,3)=0 THEN 2130
2120 O1=I1 OR I2 OR I3:GOTO2140
2130 O1=I1 OR I2
2140 S(O(I,1))=O1:RETURN
2198 '
2199 '                    --- NOR GATE ---
2200 DATA NOR,3,1
2210 GOSUB2110:S(O(I,1))=NOT(O1):RETURN
2298 '
2299 '                    --- NAND GATE ---
2300 DATA NAND,3,1
2310 GOSUB2000:S(O(I,1))=NOT(O1):RETURN
2398 '
2399 '              --- EXCLUSIVE OR GATE ---
2400 DATA XOR,2,1
2410 O1=-1
2420 IFI1=-1ANDI2=-1THENO1=0
2430 IFI1=0ANDI2=0THENO1=0
2440 S(O(I,1))=O1:RETURN
2498 '
2499 '                --- INVERTER GATE ---
2500 DATA INV,1,1
2510 S(O(I,1))=NOT(I1):RETURN
2598 '
2599 '                 --- GROUND GATE ---
2600 DATA GND,0,1
2610 S(O(I,1))=0:RETURN
2698 '
2699 '            --- J-K FLIP-FLOP GATE ---
2700 DATA J-KFF,3,2
2710 O1=S(O(I,1)):IFI3=0THEN2750
2720 IFI1=-1ANDI2=-1THENO1=NOT(O1)
2730 IFI1=-1ANDI2=0THENO1=-1
2740 IFI1=0ANDI2=-1THENO1=0
2750 S(O(I,1))=O1:S(O(I,2))=NOT(O1):RETURN
2798 '
2799 '              --- FULL ADDER GATE ---
2800 DATA FADD,3,2
2810 O1=0:O2=0:SUM=I1+I2+I3
2820 IFSUM=-1THENO1=-1
2830 IFSUM=-2THENO2=-1
2840 IFSUM=-3THENO1=-1:O2=-1
2850 S(O(I,1))=O1:S(O(I,2))=O2:RETURN
2898 '
2899 '            --- DATA END MARKER ---
2900 DATA END
```

Screen Status Byte

by Arthur R. Jackman

The screen status byte is located at memory location 16445 decimal, or 403D hexadecimal. You can read the value of the status byte in BASIC by using the statement:

```
PRINT PEEK(16445)
```

To read the value in any assembly-language program, use:

```
STATUS  EQU  403DH
        LD   A,(STATUS)
```

The screen status byte usually contains a zero value. When the command PRINT CHR$(23) is executed, and the screen is put into the 32-character format, the status byte contains a value of 8. This might not seem like much at first, but there are several occasions when this information can be very useful.

When I was developing a screen graphics print dump routine for my Axiom printer, I developed two calling sequences, one for standard 64-character mode and another for the wide 32-character mode. After discovering the status byte, I was able to build in a test for the wide character mode and automatically output the screen to the printer in the correct format.

Another time, I added a sound routine USR call to a game that used the wide character screen format. When the sound routine was called, however, the screen display changed to the 64-character format, resulting in widely-spaced, normal-sized letters and graphics.

How to Use the Status Byte

Look at Program Listing 1, which demonstrates the effects of the usual sound routine on the screen format. The screen is put into the 32-character format. During the sound output routine, the screen is forced into the 64-character format. The screen stays in this mode until a STOP or END statement is executed. See Figures 1, 2, and 3, which are screen dumps printed on the Axiom printer.

Program Listing 2 corrects the problem by setting bit 3 (adding 8) in the value output to port 255. The screen, after being put into the 32-character format, stays in that mode during the sound output.

Program Listing 3 shows the opposite problem. The screen is initially in the 64-character mode and the sound routine from Program Listing 2 sets the output to 32-character format.

In Program Listing 4, the operator chooses the screen format, and the sound routine must sense it and output accordingly. The status byte (16445) has its bit 3 set according to the screen format just as we set bit 3 of port 255 for the proper output. Simply add the status byte value to the desired output value. This is shown in line 190 of Program Listing 4.

Control of Screen Format

If you try to force the status byte to the opposite value you have a convenient way to switch from one format to the other. We have always had PRINT CHR$(23) to set the 32-character format. Now, instead of using CLS to get back to the 64-character format, when you need to save screen data, you can get back to 64 characters by using:

```
POKE 16445,0
```

You can force the 32-character format with:

```
POKE 16445,8
```

The status byte at 16445 (403DH) contains 0 for the 64-character format and 8 for the 32-character format. You can use this to get the current screen status with PEEK(16445). You can also change the screen format by using POKE 16445,0 for the 64-character format and POKE 16445,8 for the 32-character format.

```
This is a demonstration of
wide character format.  Plug
an amplifier and speaker to
the tape out connector for
the sound output.
```

Figure 1

```
This is a demonstration of
wide character format.  Plug
an amplifier and speaker to
the tape out connector for
the sound output.

Notice what happens during
the sound output.  The screen
is put into the 64 character
format by the OUT command.
```

Figure 2

an amplifier and speaker to
the tape out connector for
the sound output.

Notice what happens during
the sound output. The screen
is put into the 64 character
format by the OUT command.

As long as the program runs
the screen stays in the 64
character format. When STOP or
END is executed the screen
returns to the 32 character
format.

Figure 3

Program Listing 1

```
100 'PROGRAM #1
110 'demonstrates the incompatibilty of wide character format
120 'with the usual sound routine
130 '
140 CLS: PRINT CHR$(23)
150 PRINT "This is a demonstration of
160 PRINT "wide character format.  Plug
170 PRINT "an amplifier and speaker to
180 PRINT "the tape out connector for
190 PRINT "the sound output.
200 '
210 'delay to read screen
220 FOR I=1 TO 600: NEXT
230 'output sound routine
240 FOR I=1 TO 100
250 OUT(255),1: OUT(255),2
260 NEXT I: OUT(255),1
270 '
280 PRINT
290 PRINT "Notice what happens during
300 PRINT "the sound output.  The screen
310 PRINT "is put into the 64 character
320 PRINT "format by the OUT command.
330 'delay to read screen
340 FOR I=1 TO 600: NEXT
350 PRINT
360 PRINT "As long as the program runs
370 PRINT "the screen stays in the 64
380 PRINT "character format.  When STOP or
390 PRINT "END is executed the screen
400 PRINT "returns to the 32 character
410 PRINT "format.
420 END
```

Program Listing 2

```
100 'PROGRAM #2
110 'demonstrates how to make sound routines
120 'compatible with 32 character format
130 '
140 CLS: PRINT CHR$(23)
150 PRINT "The screen is put into the
160 PRINT "32 character format and
170 PRINT "some sample text placed on
180 PRINT "the screen for demonstration.
190 '
200 'delay to read screen
210 FOR I=1 TO 600: NEXT
220 '
230 PRINT: PRINT "To retain the 32
240 PRINT "character format we need
250 PRINT "to add 8 to each OUT value
260 PRINT "from the previous example.
270 '
280 'delay to read screen
290 FOR I=1 TO 600: NEXT
300 '
310 'compatible sound routine
320 FOR I=1 TO 100
330 OUT(255),9: OUT(255),10
340 NEXT
350 '
360 PRINT
370 PRINT "Notice now that the screen
380 PRINT "stays in the 32 character
```

```
390 PRINT "format during the sound.
400 END
```

Program Listing 3

```
100 'PROGRAM #3
110 'demonstrates reverse incompatibility
120 'with 32 character format
130 '
140 CLS
150 PRINT "The screen is left in the 64
160 PRINT "character format and
170 PRINT "some sample text placed on
180 PRINT "the screen for demonstration.
190 '
200 'delay to read screen
210 FOR I=1 TO 600: NEXT
220 '
230 PRINT: PRINT "The 32 character format
240 PRINT "sound routine from program #2
250 PRINT "is used.
270 '
280 'delay to read screen
290 FOR I=1 TO 600: NEXT
300 '
310 'reverse incompatible sound routine
320 FOR I=1 TO 100
330 OUT(255),9: OUT(255),10
340 NEXT
350 '
360 PRINT
370 PRINT "Notice now that the screen
380 PRINT "is put into the 32 character
390 PRINT "format during the sound.
400 END
```

Program Listing 4

```
100 'PROGRAM #4
110 '
120 'demonstrates using the status byte to key which sound
130 'routine output bytes to use
140 '
150 GOTO 300
160 '
170 'combined sound routine
180 FOR I=1 TO 100
190 OUT(255),1+PEEK(16445): OUT(255),2+PEEK(16445)
200 NEXT I
210 RETURN
300 'control program
310 CLS: PRINT "press a letter key for 64 character format
320 PRINT "or a shifted letter key for 32 character format.
330 Z$=INKEY$: IF Z$="" GOTO 330
340 IF Z$ > "Z" PRINT CHR$(23)
350 PRINT@0,CHR$(31)
360 PRINT "This message is now in the
370 PRINT "screen format chosen by the
380 PRINT "operator.  Now the sound
390 PRINT "routine will output a tone
400 PRINT "and not affect the screen
410 PRINT "format.
420 GOSUB 180
430 GOTO 300
```

UTILITY

Modifying Scripsit to Send Control Characters to Printers

Shortstuff

Modifying Scripsit to Send Control Characters to Printers

by Albert Davis

It is common knowledge that Scripsit lacks the ability to support the special features of some printers. This article describes a simple modification which allows you to send all control codes to the printer. Table 1 shows the control codes. After adding the modification, you can change type fonts, print graphics, underline, print subscripts and superscripts, do reverse indentation, and even print in Japanese if your printer has these capabilities. There are two limitations: First, you must give up one character, such as &. Second, any lines that contain control characters will not justify properly. When a line containing a control code is encountered, Scripsit counts the control code as it would a normal character. However, when that line goes to the printer, the control code is not printed, and the right margin fails to justify. If you must have justified text and control characters in the same document, you must format a few lines by hand.

The process is a simple matter of intercepting the print routine, which Radio Shack tells you how to do. The modification tells Scripsit to sense this special character but not print it. Then, modify the following character and print the modified character. The modification I use adds 96 (60H) to the ASCII code and sets the most significant bit. The only purpose of this is to make modified Scripsit boundary markers print as no-ops instead of lowercase letters. With the Epson MX-80 printer, this gives access to the entire set of control characters and graphics (Table 2). Another modification is to change all carriage returns to line feeds. This means you can switch the printers' auto line feed off. Then when you send a special carriage return through the modification, the printer will double print a line.

Scripsit is easy to modify because the print routine is at the end of the program, and there is some spare memory present between the end of the program and the work space. This is more than enough room for this simple modification. The space from 7AA5H to 7C28H appears to be available; so it is not necessary to protect any memory. Since the type of printer Scripsit was designed for is still being used, the printer ready test does not need modification. The initialization routine sends a string of characters to the printer to set it back to normal so that every printout starts from the same setting.

According to Radio Shack in a sheet that accompanies Scripsit, there are five areas that need to be modified:

1) 5267H–5298H: Patch Scripsit to protect this driver. This is not necessary if you use the spare memory.

2) 5F63H–5F64H: Skip the printer ready test. This change is not necessary since the printer ready test is still used.
3) 663FH–6641H: Execute your initial routine. This is done by overwriting these locations with the modification.
4) 7A97H–7A99H: Execute your driver routine. This location prints only spaces, so you do not have to change it.
5) 7A9EH–7AA0H: Execute your driver routine. This is at the end of Scripsit and is overwritten by the modification.

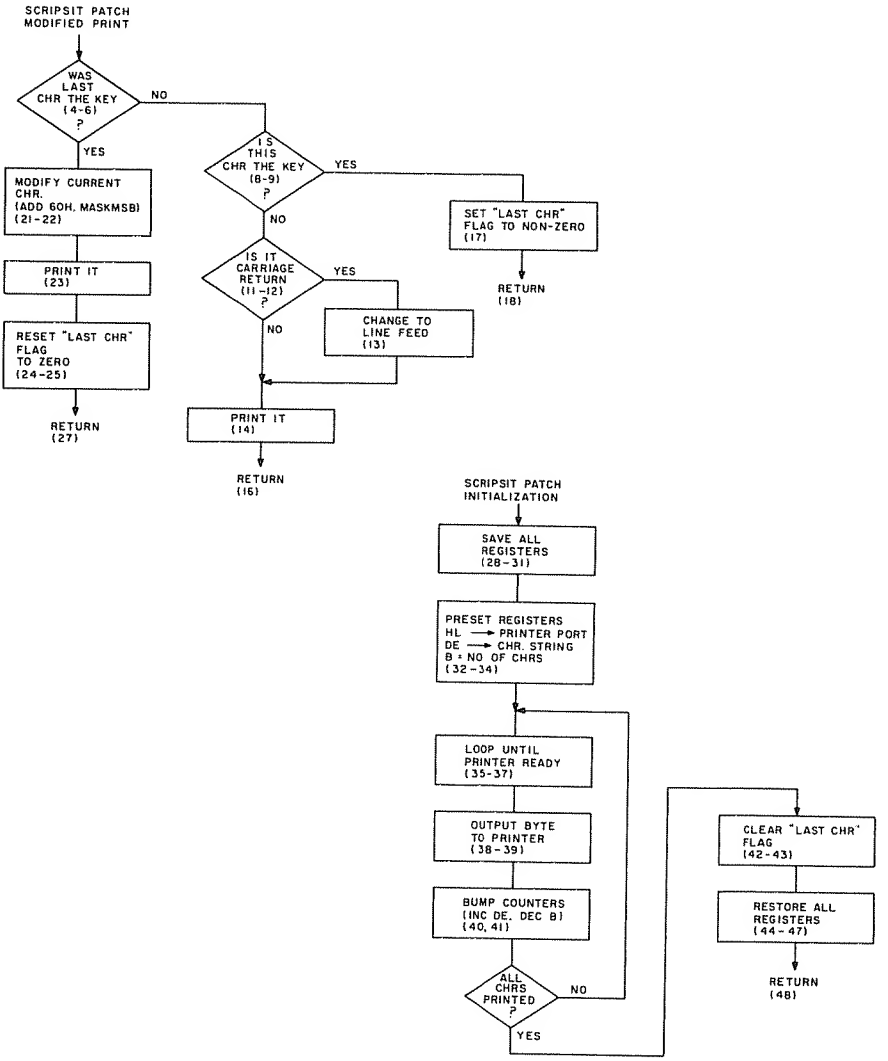


Figure 1. *Scripsit modification flowcharts*

&.	double-width on
&4	double-width off
&/	compressed on
&2	compressed off
&;E	emphasized on
&;F	emphasized off
&;G	double-strike on
&;H	double-strike off
&'	buzzer
&8	cancel unprinted data (reverse indent)
&)	horizontal tab
& +	vertical tab
&*	line feed
& –	carriage return
&,	form feed
&1	select printer
&3	de-select printer
&;0	set line spacing to 1/8 inch (nine dots)
&;1	set line spacing to 7/72 inch (seven dots)
&;2	set line spacing as set by &;A (1/6 inch, 12 dots)
&;A	preset line spacing
&;B	set vertical tabs
&;D	set horizontal tabs
&;C	set form length
&space	end tab set

Table 1. *Scripsit/Epson control codes*

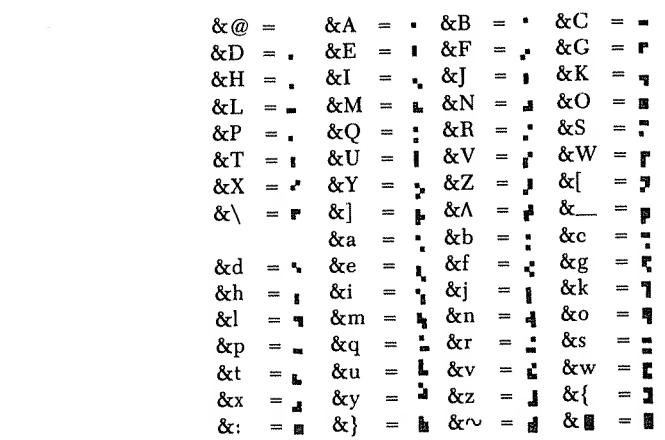


Table 2. *Scripsit/Epson control characters and graphics*

Assemble the new driver with an origin of 7A9EH. This causes it to overwrite the last few bytes of Scripsit which contain the print routine and a jump back to the rest of the program. The new print routine, therefore, must end with a JP 5F74H instruction. The initialization routine can be located anywhere in the empty space, but Scripsit must be modified to call it. In this case, we want to keep most of the existing routine, because it prints the not ready message and gives you a chance to stop the printout if the printer is not ready. The old routine sends a carriage return to the printer. This one line must be replaced by the new routine. We must change 665EH to CALL INIT by setting a second origin in the modification program at 665EH and putting the new instruction there. This must be done with either Radio Shack's tape assembler or Apparat's disk assembler. Radio Shack's disk assembler does not work for this because it fills all the memory in between with zeros, wiping out Scripsit.

Modifying Scripsit

Type in Program Listing 1, using a tape-based editor/assembler or the Apparat-modified editor/assembler, which is supplied on NEWDOS system disks. If you do not have a suitable editor/assembler, you can enter the machine code directly, using the TRSDOS DEBUG utility.

The following instructions pertain to users of Apparat's NEWDOS. If you use TRSDOS, follow the instructions in the next section. After you have entered the program listing, assemble it and save it in a disk file called SCRMOD/CMD. Then follow these steps to modify Scripsit: 1) Return the computer to the DOS command mode (DOS Ready). 2) Type LOAD SCRIPSIT/CMD and press ENTER. This loads Scripsit into memory, but does not execute it. 3) Type LOAD SCRMOD/CMD and press ENTER. This loads the modification machine code into memory. 4) Type DUMP SCRM/CMD 5200H 7AFFH 5200H and press ENTER. This dumps the newly-modified Scripsit into a file called SCRM/CMD. 5) SCRM/CMD is now the modified Scripsit, which runs normally.

If you don't have NEWDOS and if you don't mind loading two programs each time you wish to use Scripsit, follow these instructions: 1) Assemble the program and save the machine code in a disk file called SCRMOD/CMD. 2) Return to the DOS command mode (DOS Ready). 3) Type LOAD SCRIPSIT/CMD and press ENTER. 4) Type SCRMOD and press ENTER. This loads the modifying code into memory and begins execution of Scripsit.

If your disk operating system has the capability of executing DO or CHAIN files, you could set up a file that would load the above programs automatically upon booting.

Modifying with TRSDOS

There is a problem with the double loading procedure when using

TRSDOS. The TRSDOS LOAD command wipes out part of the program previously loaded, making multiple LOADs impossible. Follow this procedure, pressing ENTER after each command: 1) Assemble and save the modifying code as explained above. 2) Return to the DOS command mode (DOS Ready). 3) Type DEBUG. The computer responds with DOS Ready. 4) Type SCRIPSIT. This loads Scripsit and enters the debug utility. 5) Type G402D. This returns you to DOS Ready. 6) Type SCRM. This loads the modifying code and again enters the debug utility. 7) Type G. This executes the now-modified Scripsit.

Some Tricks

Now that Scripsit is modified to respond to control characters, there are a few tricks you can use. A carriage return (& –) returns the carriage to the stop, not to the point at which Scripsit sets the margins. This can be used for reverse indentation. Do a carriage return, then insert spaces, or tab to put the start of the text where you want it.

To do underlining, put a carriage return (& –) after the line to be underlined. Add spaces, or tab from the edge of the paper to the point to be underlined, then print the underline character.

The cancel (&8) character can be used for reverse indenting, but not for underlining or overprinting, since the printer forgets all that precedes the control code on the current line, including the spaces that Scripsit added. This works even if your printer does not overprint. The tabs can be used for formatting, or in place of spaces to set the position of the reverse indent.

With the MX-80 printer, the type changes, except double width, apply to the entire line, but this can be overcome by using a carriage return and then overprinting. Print the normal type on the first pass and the special type on the second.

If you want to cancel the carriage return and line feed from one of Scripsit's boundary markers, prefix it with &. This changes it to a character which has no effect. This is helpful when the modified characters result in a line too long for Scripsit. To get a long line, use &, then the line-boundary marker. Scripsit thinks it is a new line, so it resets the counter. The printer, however, keeps going. Be careful when you do this. It throws off Scripsit's line counter. Insert a special line feed (&*) somewhere else on the page to put the paper back where Scripsit thinks it is.

If you want superscripts, subscripts, or unusual line spacing, use the printer's control codes. If you want the printer to wake you up when it gets to a certain part, use the buzzer (&'). You are limited only by the printer. With a little imagination, you can overcome some of the printer's limitations. Almost all printers have provisions for boldface, overprinting, and more than one type font. The details here apply to the Epson MX-80 printer, but the same principles and probably the same program should work with most other printers.

Phantom Keys

There is a trick you can use to print the extra characters, such as brackets and underlining, that are not on the keyboard. The way the keyboard works allows you to do this without modifying Scripsit. If you press several keys at a time, the computer is tricked into thinking different keys were pressed, such as the ones that are not there. This, of course, produces a few spurious characters which you can delete. There are empty positions in the XYZ row. If there were keys there, they would make up the missing characters.

To get the extra characters, you must first connect the XYZ row to another row, such as HIJ. Do this by pushing the Y and I keys simultaneously. Pressing H and X also works, but is less convenient. Then, whenever another key in the HIJ row (KLMNO) is pressed, two characters are generated: the one on the key and the phantom one. If you hold them down, one character repeats. The odds of getting the phantom character to repeat are about 80 percent. Try it a few times until you get the right one. Be sure to press Y and I first, then the other key. After you have done this, delete the extra characters. This is a nuisance, but for the few times these characters are needed, it works. You can get nine extra characters this way, including underlining (YIO). Table 3 shows the phantom characters available from the MX-80. These may print different from what shows on the screen; for example, up arrow on the screen (YIK) usually prints as a left bracket.

yik	=	{	yil	=	¦	yim	=	}	yin	=	~
YIK	=	[	YIL	=	\	YIM	=	]	YIN	=	^

yio	=	
YIO	=	_

Table 3. *Scripsit phantom characters*

Other Improvements

There are two other improvements you can make. The first is to return to DOS Ready instead of rebooting. This makes Scripsit more compatible with double density systems. The other is to use the system's top of memory instead of having Scripsit find its own.

To return to DOS ready, change a JP 0000 instruction at 6594H to JP 402DH. I have taken this one step further to clear the screen. It now becomes JP EXIT, and the EXIT is CALL 01C9H (clear screen from ROM), then JP 402DH.

To use the system's top of memory, I have copied Apparat's zap. This wipes out Scripsit's routine at 5260H to find the top of memory, and replaces it with LD HL, (4049H) and a bunch of no-ops. This protects anything you have stashed at the top of memory.

Program Listing. *Scripsit patch*

Encyclopedia Loader

```
                00001 ; Scripsit patch .. Albert Davis .. 1981
7A9E            00002          ORG   7A9EH
7A9E F5         00003 DRIV     PUSH  AF         ; save chr in A
7A9F 3AEE7A     00004          LD    A,(LAST)   ; get last chr flag
7AA2 B7         00005          OR    A          ; set flags
7AA3 2019       00006          JR    NZ,SPEC    ; jump if last chr was the key
7AA5 F1         00007          POP   AF         ; get back current chr to print
7AA6 FE26       00008          CP    '&'        ; is it the key?
7AA8 280E       00009          JR    Z,IGN      ; if so, jump
7AAA F5         00010          PUSH  AF         ; save chr again
7AAB FE0D       00011          CP    0DH        ; is it carriage return?
7AAD 2002       00012          JR    NZ,NORMAL  ; jump if no
7AAF 3E0A       00013          LD    A,0AH      ; change to line feed
7AB1 32E837     00014 NORMAL   LD    (37E8H),A  ; output chr to printer
7AB4 F1         00015          POP   AF         ; restore current character
7AB5 C3745F     00016          JP    5F74H      ; return to Scripsit
7AB8 32EE7A     00017 IGN      LD    (LAST),A   ; store chr (not zero) as flag
7ABB C3745F     00018          JP    5F74H      ; don't print it, return
7ABE F1         00019 SPEC     POP   AF         ; get back current chr
7ABF F5         00020          PUSH  AF         ; save it again
7AC0 C660       00021          ADD   A,60H      ; move it up to control/graphics
7AC2 F680       00022          OR    80H        ; make sure it is > 80H
7AC4 32E837     00023          LD    (37E8H),A  ; print it (it's special now)
7AC7 AF         00024          XOR   A          ; store zero as "last" flag
7AC8 32EE7A     00025          LD    (LAST),A   ;       "
7ACB F1         00026          POP   AF         ; restore current chr
7ACC C3745F     00027          JP    5F74H      ; return to Scripsit
7ACF F5         00028 INIT     PUSH  AF         ; printer initialization routine
7AD0 E5         00029          PUSH  HL         ; save all registers
7AD1 D5         00030          PUSH  DE         ;       "
7AD2 C5         00031          PUSH  BC         ;       "
7AD3 21E837     00032          LD    HL,37E8H   ; location of printer port
7AD6 11EF7A     00033          LD    DE,STRING  ; string of chrs to "print"
7AD9 060A       00034          LD    B,0AH      ; there are 10 of them
7ADB 4E         00035 NOTRDY   LD    C,(HL)     ; check printer status
7ADC CB79       00036          BIT   7,C        ;       "
7ADE 20FB       00037          JR    NZ,NOTRDY  ; loop until ready
7AE0 1A         00038          LD    A,(DE)     ; get chr to "print"
7AE1 77         00039          LD    (HL),A     ; "print" it
7AE2 13         00040          INC   DE         ; bump chr counter
7AE3 10F6       00041          DJNZ  NOTRDY     ; loop, print the next one
7AE5 AF         00042          XOR   A          ; clear "last" flag
7AE6 32EE7A     00043          LD    (LAST),A   ;       "
7AE9 C1         00044          POP   BC         ; restore registers
7AEA D1         00045          POP   DE         ;       "
7AEB E1         00046          POP   HL         ;       "
7AEC F1         00047          POP   AF         ;       "
7AED C9         00048          RET
7AEE 00         00049 LAST     NOP              ; place to store flag
7AEF 12         00050 STRING   DEFB  12H        ; turn off compressed mode
7AF0 1B         00051          DEFB  1BH        ; "escape"
7AF1 46         00052          DEFB  'F'        ; turn off emphasized mode
7AF2 1B         00053          DEFB  1BH        ; "escape"
7AF3 48         00054          DEFB  'H'        ; turn off double strike mode
7AF4 1B         00055          DEFB  1BH        ; "escape"
7AF5 41         00056          DEFB  'A'        ; preset line spacing
7AF6 8C         00057          DEFB  8CH        ; preset it to 1/6"
7AF7 1B         00058          DEFB  1BH        ; "escape"
7AF8 32         00059          DEFB  '2'        ; set line space to 1/6"
7AF9 CDC901     00070 EXIT     CALL  01C9H      ; clear screen, home cursor
7AFC C32D40     00071          JP    402DH      ; go back to DOS
7AFF 00         00072          NOP              ; dummy byte
665E            00073          ORG   665EH      ; patch Scripsit to call new INIT
665E CDCF7A     00074          CALL  INIT       ; by overwriting one instruction
6594            00075          ORG   6594H      ; patch scripsit to return to DOS
6594 C3F97A     00076          JP    EXIT       ; instead of rebooting
5260            00077          ORG   5260H      ; patch Scripsit to use top of
```

Program continued

```
5260 2A4940   00078          LD    HL,(4049H); memory from DOS instaed of
5263 00       00079          NOP             ; figuring its own
5264 00       00080          NOP             ; wipe out old routine
5265 00       00081          NOP
5266 00       00082          NOP
5267 00       00083          NOP
5268 00       00084          NOP
5269 00       00085          NOP
5200          00086          END   5200H     ; Scripsit entry address
00000 TOTAL ERRORS
```

Shortstuff

by Roger Schrag

The Level II BASIC manual states that Level II has cassette data file capabilities built in. All you have to do is type PRINT# – 1 and a list of variables to save them on tape. To get them back, you just need to INPUT# – 1 them. However, there is a serious problem: speed.

Suppose you have a personal finance program that uses cassette data files. Also suppose that all important data is held in the array A(X) and when you are through entering your financial information, the entire contents of this array will be written onto cassette. A routine to do that might look like this:

```
FOR X = 1 TO 100:PRINT# – 1, A(X):NEXT X
```

The lack of speed in cassette data files becomes apparent when, seven minutes later, this routine is still writing data to tape.

Shortstuff is a program to allow faster data files. Program Listing 1 is the BASIC version, and Program Listing 2 is the assembly-language source code. To make a program use faster data files, you CLOAD the program, add four short lines of text, and CSAVE the new copy. You now have a stand-alone BASIC program which supports faster data files. By stand-alone I mean that every time you CLOAD the program, it will use faster data files. You will not have to load in any driver routines, or even set the memory size.

To speed up Level II's data files, first find out what slows down the process. In the example above, it takes about seven minutes and five seconds to put all the data on tape. Here is another example:

```
FOR X + 1 TO 100:PRINT# – 1, :NEXT X
```

This turns on the tape recorder, writes a leader and sync byte, then turns off the recorder. It does this 100 times. It takes about six minutes and 50 seconds to execute. The first example also turns on the recorder, writes a leader and sync byte, writes the value of a variable, then shuts off the recorder. It also does this 100 times. It takes about seven minutes and five seconds to execute.

As much as 97 percent of the time required to write a data file may be spent writing the leader, with as little as three percent of the time spent writing the actual value of the variable on tape. Of the seven minutes and five seconds required to execute the first example, only 15 seconds are spent actually writing the data.

A leader is a string of 256 zeros written to tape before the actual data. When you CSAVE a program, you write a leader first. When you PRINT# – 1 data, you also write a leader first. The string of 256 zeros is absolutely useless to the computer. When you load a program, Level II ignores all of the leader and waits for the sync byte. The sync byte synchronizes everything so that information is read from the tape exactly as it was written on the tape in the first place.

Shortstuff creates cassette data files more quickly by writing shorter leaders. When Shortstuff writes a leader, it writes 20 zeros instead of 256. Thus, 236 bytes less are written to tape for each data file. Therefore, it takes less time to write the files, less tape to store them, and less time to load them back into the computer.

When Shortstuff is initialized, it patches into the PRINT command. Whenever you type PRINT, whether it is PRINT@, PRINTTAB, PRINT# – 1, or anything else with PRINT, ROM does a call to memory location 16842. Usually an RET instruction is stored here, and control goes back into the ROM where the statement is processed further. However, Shortstuff changes this RET to a JP to the Shortstuff routine. Whenever you type in PRINT, Shortstuff gains control. It checks the character directly after the word PRINT. If this character is anything except an asterisk, control is returned to ROM and continues as usual. However, if the character after the PRINT is an asterisk, the program turns the tape recorder on, writes a short leader and sync byte, and returns control to ROM at the point where ROM processes the normal PRINT# – 1 instruction.

With Shortstuff initialized, you have a new command: PRINT*. If you type in PRINT*A$, the program will write the contents of A$ onto tape with a short leader. This also works with numeric variables, constants, and anything else that is valid with PRINT# – 1. You can also use device numbers; PRINT*# – 1, "THIS WORKS" is acceptable, and so is PRINT*# – 2, "SHORTSTUFF". Note that PRINT# – 1 and PRINT# – 2 will work normally with Shortstuff in memory.

It is simple to modify a program in BASIC to use short leaders on the data files. First CLOAD the program, then key in the program shown in the first listing, then look through the listing of your program for every location in which PRINT# – 1 is used. Add an asterisk between the PRINT and the pound sign in every case. When you have all of the PRINT# – 1 statements changed, CSAVE the new version of your program. From now on, every time you CLOAD this program, it will write out short leaders, improving the overall speed of the data files. Look at this example:

```
FOR X = 1 to 100:PRINT*# – 1, A(X):NEXT X
```

Notice that it is the same as the first example, except that it will write short leaders if Shortstuff is in memory and only takes about 51 seconds to execute. See Table 1 to adapt Shortstuff to different memory sizes.

Memory	Change M1 to	Change M2 to
4K	20455	79
32K	−16409	191
48K	−25	255

Table 1. *Changes for different memory sizes*

Shortstuff is a very simple modification which can be performed on just about any BASIC program to improve its data file efficiency considerably.

Program Listing 1. *Shortstuff, BASIC version*

Encyclopedia Loader

```
5 CLEAR 50 : POKE 16561,231 : CLEAR 50                'SET MEM SIZE
10 M1=127 : M2=32743                                  'FOR 16K MACHINE
20 FOR X=M2 TO M2+24 : READ Y : POKE X,Y : NEXT X      'POKE IN CODE
30 POKE 16842,195 : POKE 16843,231 : POKE 16844,M1     'CHANGE PRINT POINTER
40 DATA 254, 207, 192, 51, 51, 35, 205, 254, 1, 6, 25   :'MACHINE CODE TO BE
50 DATA 175, 205, 100, 2, 16, 251, 62, 165, 205, 100    :'  POKED INTO HIGH
60 DATA 2, 195, 150, 32                                 :'   MEMORY
```

Program Listing 2. *Shortstuff, assembly-language source code*

```
7FD9            00100           ORG     32729
7FD9 3EC3       00110 START     LD      A,0C3H          ;CODE FOR JP OPCODE
7FDB 32CA41     00120           LD      (41CAH),A       ;STORE AT PRINT ADDRESS
7FDE 21E77F     00130           LD      HL,SHORT        ;ADDRESS OF SHORTSTUFF
7FE1 22CB41     00140           LD      (41CBH),HL      ;JUMP TO SHORTSTUFF FOR PRINT
7FE4 C3CC06     00150           JP      6CCH            ;RETURN TO BASIC
7FE7 FECF       00160 SHORT     CP      0CFH            ;IS NEXT CHARACTER AN ASTERISK?
7FE9 C0         00170           RET     NZ              ;RETURN IF NOT
7FEA 33         00180           INC     SP              ;MAKE UP FOR CALL TO 41CAH
7FEB 33         00190           INC     SP
7FEC 23         00200           INC     HL              ;POINT TO CHARACTER AFTER ASTERISK
7FED CDFE01     00210           CALL    1FEH            ;CALL ROM ROUTINE TO SELECT RECORDER
                00220 ;========== WRITE A SHORTER LEADER ==========
7FF0 0614       00230           LD      B,14H           ;WRITE 20 ZEROS INSTEAD OF 256
7FF2 AF         00240           XOR     A               ;ZERO A REGISTER
7FF3 CD6402     00250 LOOP      CALL    264H            ;WRITE BYTE TO TAPE
7FF6 10FB       00260           DJNZ    LOOP            ;GO BACK 20 TIMES
7FF8 3EA5       00270           LD      A,0A5H          ;SYNC BYTE
7FFA CD6402     00280           CALL    264H            ;WRITE IT
7FFD C39620     00290           JP      2096H           ;RETURN TO ROM
0000            00300           END
00000 TOTAL ERRORS
```

APPENDIX

Appendix A
Appendix B
Appendix C

APPENDIX A

BASIC Program Listings

Debugging someone else's mistakes is no fun. In a business environment, where programs are continuously updated and programmers come and go, well-commented and structured programs are a must. Indeed, it behooves any serious programmer to learn structured technique.

The BASIC language has no inherent structure. Most interpreters allow remark lines and some are capable of ignoring unnecessary spacing, but BASIC is still more "Beginner's Instruction Code" than "All-purpose."

The listings in this encyclopedia are an attempt at formatting the TRS-80 BASICs. We think it makes them easier to read, easier to trace, and less imposing when it comes time to type them into the computer. You should *not*, however, type them in exactly as they appear. Follow normal syntax and entry procedures as described in your user's manual.

Level I Programs

Programs originally in Level I have been converted to allow running in Level II. To run in Level I, follow this procedure:

- Delete any dimension statements. Example: DIM A (25).
- Change PRINT@ to PRINTAT.
- Make sure that no INPUT variable is a STRING variable.
  Example: INPUT A$ would be changed to INPUT A and subsequent code made to agree.
- Abbreviate all BASIC statements as allowed by Level I.
  Example: *PRINT* is abbreviated *P.*

Model III Users

For the Model I, OUT255,0 and OUT255,4 turn the cassette motor off and on, respectively. For the Model III, change these statements to OUT236,0 and OUT236,2.

APPENDIX B

Glossary

A

access time—the elapsed time between a request for data and the appearance of valid data on the output pins of a memory chip. Usually 200–450 nanoseconds for TRS-80 RAM.

accumulator—the main register(s) in a microprocessor used for arithmetic, shifting, logical, and other operations.

accuracy—generally, the quality or freedom from mistake or error; the extent to which the results of a calculation or a measurement approach the true value of the actual quantities.

acoustic coupler—a connection to a modem allowing signals to be transmitted through a regular telephone handset.

A/D converter—analog to digital converter. See D/A converter.

address—a code that specifies a register, memory location, or other data source or destination.

ALGOL—an acronym for ALGOrithmic Language. A very high-level language used in scientific applications, generally on large-scale computers.

algorithm—a predetermined process for the solution of a problem or completion of a task in a finite number of steps.

alignment—the process of adjusting components of a system for proper interrelationships, including adjustments and synchronization for the components in a system. For the TRS-80, this usually applies to cassette heads and disk drives.

alphanumerics—refer to the letters of the alphabet and digits of the number system, specifically omitting the characters of punctuation and syntax.

alternating current—ac. Electric current that reverses direction periodically, usually many times per second.

ALU—Arithmetic Logic Unit.

analog—the representation of a physical variable by another variable insofar as the proportional relationships are the same over some specified range.

AND—a Boolean logic function. Two operators are tested and, if both are true, the answer is true. Truth is indicated by a high bit, or 1 in machine language, or a positive value in BASIC. If the operators are bytes or words, each element is tested separately. A bit-by-bit logical operation which produces a one in the result bit only if both operand bits are ones.

anode—in a semiconductor diode, the terminal toward which electrons flow from an external circuit; the positive terminal.

APL—A Programming Language; a popular and powerful high-level mathematical language with extensive symbol manipulation.

argument—any of the independent variables accompanying a command.

Arithmetic Logic Unit—ALU. The section of a microprocessor which performs arithmetic functions such as addition or subtraction and logic functions such as ANDing.

array—a collection of data items arranged in a meaningful pattern such as rows and columns which allow the collection and retrieval of data.

ASCII—American Standard Code for Information Interchange. An almost universally accepted code (at least for punctuation and capital letters) where characters and printer commands are represented by numbers between 0 and 255 (base 10). The number is referred to as an ASCII code.

assembler—software that translates operational codes into their binary equivalents on a statement-for-statement basis.

assembly language—a symbolic computer language that is translated by an assembler program into machine language, the numeric codes that are equivalent to microprocessor instructions.

B

backup—1) refers to making copies of all software and data stored externally; 2) having duplicate hardware available.

base—the starting point for representation of a number in written form, where numbers are expressed as multiples of powers of the base value.

BASIC—an acronym for Beginner's All-purpose Symbolic Instruction Code. Developed at Dartmouth College and similar to FORTRAN. The standard, high-level, interactive language for microcomputers.

batch processing—a method of computing in which many of the same types of jobs or programs are done in one machine run. For example, a programming class may type programs on data cards and turn them over to the computer operator. All the cards are put into the card reader, and the results of each person's program are returned later. This is contrasted with interactive computing.

baud—1) a unit of data transmission speed equal to the number of code elements (bits) per second; 2) a unit of signaling speed equal to the number of discrete conditions or signal events per second.

baud rate—a measure of the speed at which serial data is transmitted electronically. The equivalent of bits per second (bps) in microcomputing.

benchmark—to test performance against a known standard.

BCD—binary coded decimal. The 4-bit binary notation in which individual decimal digits (0 through 9) are represented by 4-bit binary numerals; e.g., the number 23 is represented by 0010 0011 in the BCD notation.

bias—a dc voltage applied to a transistor control electrode to establish the desired operating point.

bidirectional bus—a bus structure used for the two-way transmission of signals, that is, both input and output.

bidirectional printer—a printer capable of printing both left-to-right and right-to-left. Data is prestored in a fixed-size buffer.

binary—a number system which uses only 0 and 1 as digits. It is the equivalent of base 2. Used in microcomputing because it is easy to represent 1s and 0s by high and low electrical signals.

binary digit—the two digits, 0 and 1, used in binary notation. Often shortened to bit.

bi-stable—two-state

bit—an abbreviation for binary digit. A 0 or 1 in the binary number system. A single high or low signal in a computer.

bit position—the position of a binary digit within a byte or larger group of binary digits. Bit positions in the Model I, II, III, and Color Computer are numbered from right to left, zero through N. This number corresponds to the power of two represented.

Boolean algebra—a mathematical system of logic first identified by George Boole, a 19th century English mathematician. Routines are described by combinations of ANDs, ORs, XORs, NOTs, and IF-THENs. All computer functions are based upon these operators.

boot—short for bootstrap loader or the use of one. The bootstrap loader is a very short routine whose purpose is to load a more sophisticated system into the computer when it is first turned on. On some machines it is keyed in, and on others it is in read only memory (ROM). Using this program is called booting or cold-starting the system.

bps—bits per second.

buffer—memory set aside temporarily for use by the program. Particularly refers to memory used to make up differences in the data transfer rates of the computer and external devices such as printers and disks.

bug—an error in software or hardware.

bus—an ordered collection of all address, data, timing, and status lines in the computer.

byte—eight bits that are read simultaneously as a single code.

C

CAI—an acronym for Computer Aided Instruction.

card—a specially designed sheet of cardboard with holes punched in specific columns. The placement of the holes represents machine-readable data. Also a term referring to a printed circuit board.

card reader—a device for reading information from punched cards.

cassette recorder—a magnetic tape recording and playback device for entering or storing programs.

cathode—in a semiconductor diode, the terminal from which electrons flow to an external circuit; the negative terminal.

character—a single symbol that is represented inside the computer by a specific code.

checksum—a method of detecting errors in a block of data by adding each piece of data in the block to a sum and comparing the final result to a predetermined result for the block of data.

chip—the shaped and processed semiconductor die mounted on a substrate to form a transistor or other semiconductor device.

circuit—a conductor or system of conductors through which an electric current may flow.

circuit card—a printed circuit board containing electronic components.

clear—to return a memory to a non-programmed state, usually represented as 0 or OFF (empty).

clock—a simple circuit that generates the synchronization signals for the microprocessor. The speed or frequency of this clock directly affects the speed at which the computer can perform, regardless of the speed of which the individual chips are capable.

COBOL—COmmon Business-Oriented Language. A language used primarily for data processing. Allows programming statements that are very similar to English sentences.

compiler—software that will convert a program written in a high-level language to binary code, on a many-for-one basis.

complement—a mathematical calculation. In computers it specifically refers to inverting a binary number. Any 1 is replaced by a 0, and vice versa.

computer interface—a device designed for data communication between a central computer and another unit such as a programmable controller processor.

concatenate—to put two things, each complete by itself, together to make a larger complete thing. In computers this refers to strings of characters or programs.

conductor—a substance, body, or other medium that is suitable to carry an electric current.

constant—a value that doesn't change.

CPU—central processing unit. The circuitry that actually performs the functions of the instruction set.

CRT—cathode ray tube. In computing this is just the screen the data appears on. A TV has a CRT.

cue—refers to positioning the tape on a cassette unit so that it is set up to a read/write section of tape.

cursor—a visual movable pointer used on a CRT by the programmer to indicate where an instruction is to be added to the program. The cursor is also used during editing functions.

cycle—a specific period of time, marked in the computer by the clock.

D

D/A converter—digital to analog converter. Common in interfacing computers to the outside world.

daisy wheel—a printer type which has a splined character wheel.

data—general term for numbers, letters, symbols, and analog quantities that serve as information for computer processing.

data base—refers to a series of programs each having a different function, but all using the same data. The data is stored in one location or file and each program uses it in a fashion that still allows the other program to use it.

data entry—the practice of entering data into the computer or onto a storage device. Knowledge of operating or programming a computer is not necessary for a data entry operator.

debug—to remove bugs from a program.

decrement—to decrease the value of a number. In computers the number is in memory or a register, and the amount it is decremented is usually one.

dedicated—in computer terminology, a system set up to perform a single task.

default—that which is assumed if no specific information is given.

degauss—to demagnetize. Must be done periodically to tape and disk heads for reliable data transfer.

diagnostic program—a test program to help isolate hardware malfunctions in the programmable controller and application equipment.

digital—the representation of data in binary code. In microcomputers, a high electrical signal is a 1 and a low signal is a 0.

digital circuit—an electronic network designed to respond at input voltages at one level, and similarly, to produce output voltages at one level.

diode—a device with an anode and a cathode which permits current flow in one direction and inhibits current flow in the other direction.

direct current—dc. Electric current which flows in only one direction; the term designates a practically non-pulsating current.

disassembly—remaking an assembly source program from a machine-code program.

disk—an oxide-coated, circular, flat object, in a variety of sizes and containers, on which computer data can be stored.

disk controller—an interface between the computer and the disk drive.

disk drive—a piece of hardware that rotates the disk and performs data transfer to and from the disk.

disk operating system—DOS. The system software that manipulates the data to be sent to the disk controller.

dividend—the number that is divided by the divisor. In A/B, A is the dividend.

divisor—the number that "goes into" the dividend in a divide operation. In A/B, B is the divisor.

DMA—direct memory access. A process where the CPU is disabled or

bypassed temporarily and memory is read or written to directly.

documentation—a collection of written instructions necessary to use a piece of hardware, software, or a system.

dot-matrix printer—instead of each letter having a separate type head (like a typewriter), a single print head makes the characters by printing groups of dots. The print is not as easy to read, but such printers are less expensive to manufacture.

downtime—the time when a system is not available for production due to required maintenance.

driver—a small piece of system software used to control an external device such as a keyboard or printer.

dump—to write data from memory to an external storage device.

duplex—refers to two-way communications taking place independently, but simultaneously.

dynamic memory—circuits that require a periodic (every few milliseconds) recharge so that the stored data is not lost.

E

EAROM—an acronym for Electrical Alterable Read Only Memory. The chip can be read at normal speed, but must be written to with a slower process. Once written to, it is used like a ROM, but can be completely erased if necessary.

editor—a program that allows text to be entered into memory. Interactive languages usually have their own editors.

EOF—End Of File.

EOL—End Of Line (of text).

EPROM—Erasable Programmable Read Only Memory. A read only memory in which stored data can be erased by ultraviolet light or other means and reprogrammed bit-by-bit with appropriate voltage pulses.

Exclusive OR—a bit-by-bit logical operation which produces a one bit in the

result only if one or the other (but not both) operand bits is a one.

execution—the performance of a specific operation such as would be accomplished through processing one instruction, a series of instructions, or a complete program.

execution cycle—a cycle during which a single instruction of one specific operation is performed.

execution time—the total time required for the execution to actually occur.

expansion interface—a device attached to the computer that allows a greater amount of memory or attachment of other peripherals.

exponent—the power to which a floating-point number is raised.

F

fetch cycle—a cycle during which the next instruction to be performed is read from memory.

field-effect transistor—FET. A transistor in which the resistance of the current path from the source to drain is modulated by applying a transverse electric field between grid or gate electrodes; the electric field varies the thickness of depletion layers between the gates, thereby reducing the conductance.

file—a set of data, specifically arranged, that is treated as a single entity by the software or storage device.

firmware—software that is made semi-permanent by putting it into some type of ROM.

flag—a single bit that is high (set) or low (reset), used to indicate whether or not certain conditions exist or have occurred.

flip-flop—a bi-stable device that assumes either of two possible states such that the transition between the states must be accomplished by electronic switching.

floating-point number—a standard way of representing any size number in computers. Floating-point numbers contain a fractional portion (mantissa) and power of two (exponent) in a form similar to scientific notation.

flowcharting—a method of graphically displaying program steps, used to develop and define an algorithm before writing the actual code.

FORTRAN—FORmula TRANslator. One of the first high-level languages, written specifically to allow easy entry of mathematical problems.

full duplex—a mode of data transmission that is the equivalent of two paths—one in each direction simultaneously.

G

game theory—see von Neumann.

garbage—computer term for useless data.

gate—a circuit that performs a single Boolean function. A circuit having an output and a multiplicity of inputs, so designed that the output is energized only when a certain combination of pulses is present at the inputs.

GIGO—Garbage In, Garbage Out. One of the rules of computing. If the data going into the computer is bad, the data coming out will be bad also.

graphics—information displayed pictorially as opposed to alphanumerically.

ground—a conducting path, intentional or accidental, between an electric circuit or equipment and the earth, or some conducting body serving in place of the earth.

H

H—a suffix for hexadecimal, e.g., 4FFFH.

half duplex—data can flow in both directions, but not simultaneously. See duplex.

handshaking—a term used in data transfer. Indicates that beside the data lines there are also signal lines so both devices know precisely when to send or receive data. Handshaking requires clocking pulses on both ends of the communications line. Contrast with buffer.

hard copy—a printout; any form of printed document such as a ladder diagram, program listing, paper tape, or punched cards.

hardware—refers to any physical piece of equipment in a computer system.

hex—hexadecimal.

hexadecimal—representation of numbers in base sixteen by use of the hexadecimal digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

high—a signal line logic level. The computer senses this level and treats it as a binary 1.

high-level language—a programming language which is CPU-independent and closely resembles English.

high order—see most significant bit.

HIT—acronym for Hash Index Table. A section of the directory on a TRS-80 disk.

human engineering—usually refers to designing hardware and software with ease of use in mind.

I

IC—integrated circuit.

immediate—addressing mode in which the address of the information that an operation is supposed to act upon immediately follows the operation code.

increment—to increase, usually by one. See decrement.

indexed—addressing mode where the information is addressed by a specified value, or by the value in a specified register.

indirect—addressing mode in which the address given points to another address, and the second address is where the information actually is.

input devices—devices such as limit switches, pressure switches, push buttons, etc., that supply data to a programmable controller. These discrete inputs are two types: those with common return, and those with individual returns (referred to as isolated inputs). Other inputs include analog devices and digital encoders.

instruction—a command or order that will cause a computer to perform one particular operation.

integer variable—a BASIC variable type. It can hold values of −32,768 through +32,767 in two-byte two's complement notation.

integrated circuit—IC. An interconnected array of active and passive elements integrated with a single semiconductor substrate or deposited on the substrate and capable of performing at least one electronic circuit function. See chip.

intelligent terminal—a terminal with a CPU and a certain amount of memory that can organize the data it receives and thus achieve a high level of handshaking with the host computer.

interactive computing—refers to the appearance of a one-to-one human-computer relationship.

interface—a piece of hardware, specifically designed to hook two other devices together. Usually some software is also required.

interpreter—a piece of system software that executes a program written in a high-level language directly. While useful for interactive computing, this system is too slow for most serious programming. Contrast with compiler.

interrupt—a signal that tells the CPU that a task must be done immediately. The registers are pushed to the stack, and a routine for the interrupt is branched to. When finished, the registers are popped from the stack and the main program continues.

I/O—acronym for input/output. Refers to the transfer of data.

iteration—one pass through a given set of instructions.

J

jack—a socket, usually mounted on a device, which will receive a plug (generally mounted on a wire).

K

K—abbreviation for kilo. In computer terms 1024, in loose terms 1000.

L

language—a set of symbols and rules for representing and communicating information (data) among people, or between people and machines.

large scale integration—LSI. Any integrated circuit which has more than 100 equivalent gates manufactured simultaneously on a single slice of semiconductor material.

least significant bit—the rightmost bit in a binary value, representing 2^0.

least significant byte—refers to the lowest position digit of a number. The rightmost byte of a number or character string.

LIFO—acronym for Last In First Out. Most CPUs maintain a "stack" of memory. The last data pushed onto the stack is the first popped out.

light emitting diode—LED. A semiconductor diode that displays alphanumeric characters when supplied with a specified voltage.

light pen—a device that senses light, interfaced to the computer for the purpose of drawing on the CRT screen.

line—in communications, describes cables, telephone lines, etc., over which data is transmitted to and received from the terminal.

line printer—a high-speed printing device that prints an entire line at one time.

location—a storage position in memory.

logic—a means of solving complex problems through the repeated use of simple functions which define basic concepts. Three basic logic functions are AND, OR, and NOT.

logic diagram—a drawing which represents the logic functions AND, OR, NOT, etc.

logic level—the voltage magnitude associated with signal pulses representing ones and zeroes (1s and 0s) in binary computation.

logical shift—a type of shift in which an operand is shifted right or left, with a zero filling the vacated bit position.

loop—a set of instructions that executes itself continuously. If the programmer has the presence of mind to provide for a test, the loop is discontinued when the test is met, otherwise it goes on until the machine is shut down.

loop counter—one way to test a loop. The counter is incremented at each pass through the loop. When it reaches a certain value, the loop is terminated.

low—a logic signal voltage. The computer senses this as a binary 0.

lsb—see least significant bit.

LSI—acronym for Large Scale Integration. An integrated circuit with a large number of circuits such as a CPU. See chip.

M

machine code—refers to programming instructions that are stored in binary and can be executed directly by the CPU without any compilation, interpretation, or assembly.

machine language—the primary instructions that were designed into the CPU by the manufacturer. These instructions move data between memory and registers, perform simple adding in registers, and allow branching based on values in registers.

macro—a routine that can be separately programmed, given a name, and executed from another program. The macro can perform functions on variables in the program that called it without disturbing anything else and then return control to the calling program.

mainframe—refers to the CPU of a computer. This term is usually confined to larger computers.

mantissa—the fractional portion of a floating-point number.

matrix—a two-dimensional array of circuit elements, such as wires, diodes, etc., which can transform a digital code from one type to another.

memory—the hardware that stores data for use by the CPU. Each piece of

data (bit) is represented by some type of electrical charge. Memory can be anything from tiny magnetic doughnuts to bubbles in a fluid. Most microcomputers have chips that contain many microscopic capacitors, each capable of storing a tiny electrical charge.

metal oxide semiconductor—MOS. A metal insulator semiconductor structure in which the insulating layer is an oxide of the substrate material; for a silicon substrate the insulator is silicon oxide.

micro electronics—refers to circuits built from miniaturized components and includes integrated circuits.

microprocessor—an electronic computer processor section implemented in relatively few IC chips (typically LSI) which contain arithmetic, logic, register, control, and memory functions.

microsecond—us. One millionth of a second: 1×10^{-6} or 0.000001 second.

millisecond—ms. One thousandth of a second: 10^{-3} or 0.001 second.

minuend—the number from which the subtrahend is subtracted.

mixed number—a number consisting of an integer and fraction as, for example, 4.35 or (binary) 1010.1011.

mnemonic—a short, alphanumeric abbreviation used to represent a machine-language code. An assembler will take a program written in these mnemonics and convert it to machine code.

modem—MOdulator/DEModulator. An I/O device that allows communication over telephone lines.

module—an interchangeable plug-in item containing electronic components which may be combined with other interchangeable items to form a complete unit.

monitor—1) a CRT; 2) a short program that displays the contents of registers and memory locations and allows them to be changed. Monitors can also allow another program to execute one instruction at a time, saving programs and disassembling them.

MOS—see metal oxide semiconductor.

MOSFET—metal oxide semiconductor field effect transistor.

most significant bit—the leftmost bit in a binary value, representing the highest-order power of two. In two's complement notation, this bit is the sign bit.

most significant byte—the highest-order byte. In the multiple-precision number A13EF122H, A1H is the most significant byte.

msb—see most significant byte.

multiplexing—a method allowing several sets of data to be sent at different times over the same communication lines, yet all of the data can be used simultaneously after the final set is received. For example, several LED displays, each requiring four data lines, can all be written to with only one group of four data lines. The same concept is used with communication lines.

multiplicand—the number to be multiplied by the multiplier.

multiplier—the number that is multiplied against the multiplicand. The number "on the bottom."

N

NAND—an acronym for NOT AND. A Boolean logic expression. AND is performed, then NOT is performed to the result.

nanosecond—one billionth of a second.

nesting—putting one loop inside another. Some computers limit the number of loops that can be nested.

noise—extraneous signals; any disturbance which causes interference with the desired signal or operation.

non-volatile memory—a memory that does not lose its information while its power supply is turned off.

NOT—a Boolean operator that reverses outputs (1 becomes 0, 0 becomes 1). This is the one's complement.

O

object code—all of the machine code that is generated by a compiler or assembler. Once object code is loaded into memory it is called machine code.

octal—refers to the base 8 number system, using digits 0–7.

OEM—Original Equipment Manufacturer.

off-line—describes equipment or devices which are not connected to the communications line.

off-the-shelf—a term referring to software. A generalized program that can be used by many computer owners. It is mass produced and can be bought off-the-shelf.

on-line—a term describing a situation where one computer is connected to another, with full handshake, over a modem line.

operands—the numeric values used in the add, subtract, or other operation.

OR—a Boolean logic function. If at least one of the lines tested is high (binary 1), the answer is high.

output—the current, voltage, power, driving force, or information which a circuit or a device delivers. The terminals or other places where a circuit or device can deliver energy.

output devices—devices such as solenoids, motor starters, etc., that receive data from the programmable controller.

overflow—a condition that exists when the result of an add, subtract, or other arithmetic operation is too large to be held in the number of bits allotted.

overlay—a method of decreasing the amount of memory a program uses by allowing sections that are not in use simultaneously to load into the same area of memory. The new routine destroys the first routine, but it can always be loaded again if needed. Usually used in system programs.

oxide—an iron compound coating on tapes and disks that allows them to be magnetized so that they can be read by electrical devices and the information converted back to machine code.

P

page—refers to a 256 (2 to the 8th power) word block of memory. How large a word depends on the computer. Most micros are eight-bit word machines. Many chips do special indexed and offset addressing on the page where the program counter is pointing and/or on the first page of memory.

parallel—describes a method of data transfer where each bit of a word has its own data line, and all are transferred simultaneously. Contrast with serial.

parameter—a variable or constant that can be defined by the user and usually has a default value.

parity—a method of checking accuracy. The parity is found by adding all the bits of a word together. If the answer is even, the parity is 0 or even. If odd, the parity is 1 or odd. The bit sometimes replaces the most significant bit and usually sets a flag.

parity bit—an additional bit added to a memory word to make the sum of the number of 1s in a word always even or odd as required.

parity check—a check that tests whether the number of 1s in an array of binary digits is odd or even.

PC board—see printed circuit board.

peripheral devices—a generic term for equipment attached to a computer, such as keyboards, disk drives, cassette tapes, printers, plotters, speech synthesizers.

permutation—arrangements of things in definite order. Two binary digits have four permutations: 00, 01, 10, and 11.

PILOT—a simple language for handling English sentences and strings of alphanumeric characters. Generally used for CAI.

PL/1—an acronym for Programming Language 1. A programming language used by very large computers. It incorporates most of the better features from other programming languages. Its power comes from the fact that bits can be manipulated from the high-level language.

plotter—a device that can draw graphs and curves and is controlled by the computer through an interface.

port—a single addressable channel used for communications.

positional notation—representation of a number where each digit position represents an increasingly higher power of the base.

precision—the number of significant digits that a variable or number format may contain.

printed circuit board—a piece of plastic board with lines of a conductive material deposited on it to connect the components. The lines act like wires. These can be manufactured quickly and are easy to assemble the components on.

processor—a unit in the programmable controller which scans all the inputs and outputs in a predetermined order. The processor monitors the status of the inputs and outputs in response to the user-programmed instructions in memory, and it energizes or de-energizes outputs as a result of the logical comparisons made through these instructions.

product—the result of a multiply.

program—a sequence of instructions to be executed by the processor to control a machine or process.

PROM—Programmable Read Only Memory. A memory device that is written to once and from then on acts like a ROM.

pseudo code—a mnemonic used by assemblers that is not a command to the CPU, but a command to the assembler itself.

punched-card equipment—peripheral devices that enable punching or reading paper punched cards that hold character or binary data.

Q

quotient—the result of a divide operation.

R

RAM—acronym for Random Access Memory. An addressable LSI device used to store information in microscopic flip-flops or capacitors. Each may be set to an ON or OFF state, representing logical 1 or 0. This type of

memory is volatile, that is to say, memory is lost while power is off, unless battery backup is used.

read—to sense the presence of information in some type of storage, which includes RAM memory, magnetic tape, punched tape, etc.

real time clock—a clock in the sense that we normally think of one, interfaced to the computer.

record—a file is divided into records, each of which is organized in the same manner.

register—a fast-access memory location in the microprocessor. Used for holding intermediate results and for computation in machine language.

relative addressing—an address that is dependent upon where the program counter is presently pointing.

remainder—the amount of dividend remaining after a divide has been completed.

ROM—an acronym for Read Only Memory. Memory that is addressed by the bus, but can only be read from. If you tell the CPU to write to it, the machine will try, but the data is not remembered.

rounding—the process of truncating bits to the right of a bit position and adding zero or one to the next higher bit position based on the value to the right. Rounding the binary fraction 1011.1011 to two fractional bits, for example, results in 1011.11.

RPG—an acronym for Report Program Generator. A language for business that primarily reads data from cards and prints reports containing that data.

RS-232—an interface that converts parallel data to serial data for communications purposes. The output is universally standard.

S

scaling—multiplying a number by a fixed amount so that a fraction can be processed as an integer value.

scientific notation—a standard form for representing any size number by a mantissa and power of ten.

semiconductor—a compound that can be made to vary its resistance to electricity by mixing it differently. Layers of this material can be used to make circuits that do the same things tubes do, but using much less electricity. Transistors and integrated circuits are made from semiconductive material and are called semiconductors.

serial—a way of sending data, one bit at a time, between two devices. The bits are rejoined into bytes by the receiving device. Contrast with parallel.

sign bit—sometimes the most significant bit is used to indicate the sign of the number it represents. 1 is negative (−) and 0 is positive (+).

signed numbers—numbers that may be either positive or negative.

significant bits—the number of bits in a binary value after leading zeros have been removed.

significant digit—a digit that contributes to the precision of a number. The number of significant digits is counted beginning with the digit contributing the most value, called the most significant digit, and ending with one contributing the least value, called the least significant digit.

simulator—a computer that is programmed to mimic the action and functions of another piece of machinery, usually for training purposes. A computer is usually employed because it is cheaper to have the computer simulate these actions than to use the real thing. Airplane and power plant trainers are excellent examples.

software—refers to the programs that can be run on a computer.

solid state devices (semiconductors)—electronic components that control electron flow through solid materials such as crystals; e.g., transistors, diodes, integrated circuits.

source program—the program written in a language or mnemonics that is converted to machine code. The source program as well as the object code generated from it can be saved in mass storage devices.

SPOOL—acronym for Simultaneous Peripheral Output, On-Line. Used to overlap processing, typically, with printing.

stack—an area of memory used by the CPU and the programmer particularly for storage of register values during interrupt routines. See LIFO.

stepper motor—a special motor in a disk drive that moves the read/write head a specific distance each time power is applied. That distance defines the tracks on a disk.

storage—see memory.

subroutine—a routine within a program that ends with an instruction to return program flow to where it was before the routine began. This routine is used many times from many different places in the program, and the subroutine allows you to write the code for that routine only once. Similar to a macro.

subtrahend—the number that is subtracted from the minuend.

syntax—the term is used exactly as it is used in English composition. Every language has its own syntax.

system—a collection of units combined to work as a larger integrated unit having the capabilities of all the separate units.

system software—software that the computer must have loaded and running to work properly.

T

table—an ordered collection of variables and/or values, indexed in such a way that finding a particular one can be done quickly.

tape reader—a unit which is capable of sensing data from punched tape.

TeletypeTM—a peripheral electromechanical device for entering or outputting a program or data in either a punched paper tape or printed format.

text editor—see word processor.

time sharing—refers to systems which allow several people to use the computer at the same time.

track—a concentric area on a disk where data is stored in microscopic magnetized areas.

transistor—an active component of an electronic circuit consisting of a small

block of semiconducting material to which at least three electrical contacts are made, usually two closely spaced rectifying contacts and one ohmic (non-rectifying) contact; it may be used as an amplifier, detector, or switch.

transistor-transistor logic—TTL. A logic circuit containing two transistors, for driving large output capacitances at high speed. A family of integrated circuit logic. (Usually 5 volts is high or 1, and 0 volts is low or 0; 5V = 1, 0V = 0).

truncation—the process of dropping bits to the right of a bit position. Truncating the binary fraction 1011.1011 to a number with fraction of two bits, for example, results in 1011.10.

truth table—a table defining the results for several different variables and containing all possible states of the variables.

TTL—see transistor-transistor logic.

TTY—an abbreviation for Teletype.

two's complement—a standard way of representing positive and negative numbers in microcomputers.

U

unsigned numbers—numbers that may be only positive; absolute numbers.

utility—a program designed to aid the programmer in developing other software.

UV erasable PROM—an ultraviolet erasable PROM is a programmable read-only memory which can be cleared (set to 0) by exposure to intense ultraviolet light. After being cleared, it may be reprogrammed.

V

variable—a labeled entity that can take on any value.

volatile memory—a memory that loses its information if the power is removed from it.

von Neumann, John (1903–1957)—mathemetician. He put the concept of games, winning strategy, and different types of games into mathematical formulae. He also advanced the concept of storing the program in memory as opposed to having it on tape.

W

weighted value—the numerical value assigned to any single bit as a function of its position in the code word.

word—a grouping or a number of bits in a sequence that is treated as a unit and is stored in one memory location. If the CPU works with 8 bits, then the word length is 8 bits. Common word sizes are 4, 8, 12, 16, and 32. Some are as large as 128 bits.

word processor—a computer system dedicated to editing text and printing it in various controllable formats. See editor.

write—to store in memory or on a mass storage device.

X

XOR—a Boolean function. Acronym for eXclusive OR. Similar to OR but answer is high (1) if and only if one line is high.

Z

zero flag—a bit in the microprocessor used to record the zero/non-zero status of the result of a machine-language instruction.

zero page—refers to the first page of memory.

APPENDIX C

Since the beginning of the *Encyclopedia for the TRS-80* series, we have realized a few problems with some of the articles.

Volume 1

Page 167. Line 140 should read:

```
140 CLS : U = -1 : V = 704 : W = 64 : FOR N = 1 TO 11 : U = U + 1 :
    V = V - 64 : PRINT@V, U;: NEXT N
```

Line 145 should read:

```
145 H = 69 : I = 123 : FOR N = 1 TO 9 : FOR X = H TO I :
    PRINT@X,"-";: NEXT X : H = H + 64 : I = I + 64 : NEXT N
```

Page 182. The five lines of BASIC listing which appear at the end of the Program Listing are not a part of that program and should not be there.

Volume 2

Page 126. Table 2, three lines from the bottom. The line should read: Some TRS-80s require Shift ↓ Z.

Page 136. Line 460 should read: IF P > LL. . . (not L).

Page 143. In line 1560, remove the -128.

Page 145. Line 1800 should read:

```
1800 FOR K = 1 TO X : X$(J) = X$(J) + CHR$(ASC(MID$(A$(I),K,1)) + 128) : NEXT K
```

Page 171. Line 1070 should read:

```
1070 FOR G = F TO A - 1:IF AS$(G) OR CS$ = C$(G)
CLS:GOSUB 1250:ELSE NEXT G:GOTO 1090
```

Line 1080 should read:

```
1080 FOR F = G TO A - 1:IF AS$(F) OR CS$ = C$(F) GOSUB
1270:NEXT F:GOTO 1100:ELSE NEXT F:GOTO 1100
```

Volume 3

Page 63. Line 420 is missing. It should read:

```
420 IF Q = 0 THEN 440
```

Volume 4

Page 172. Text is missing at the top of the page. It should read:

First you open a sequential file, DISKDATA, for output onto the disk, using buffer number 1. Since there is no point in having a listing for this disk directory, it's made invisible.

```
60 OPEN "0",1,"DISKDATA:X"
70 CMD"ATTRIB DISKDATA:X (I)"
```

In the above lines, X is the designated disk drive. Note: This may not be possible. . .

Volume 8

Page 17. Line 1520 should read:

```
1520 PRINT:PRINT:PRINT"TO ADD OR DELETE STOCKS: 1) LIST
LINES 160-290. *** 2) INSERT (IN ALPHABETICAL ORDER, IF
DESIRED) OR DELETE, DATA LINES.***"
```

Line 1525 should read:

```
1525 PRINT"3) REVISE CORRESPONDING DATA IN LINES 390 AND
400. *** 4) CHANGE 'NUM' IN LINE 100 TO EQUAL NEW NUMBER
OF STOCKS."
```

Page 19. Line 3330 should read:

```
3330 IF H < > 1 THEN 90
```

Page 20. Line 3450 should read:

```
3450 GOTO 90
```

CONTENTS

Volumes 1–10

VOLUME 5

VOLUME 6

VOLUME 7

VOLUME 8

VOLUME 9

VOLUME 10

***Programs on Encyclopedia Loader™**

INDEX

Volume 10
Volumes 1–10

INDEX

INDEX COMPILED BY NAN MCCARTHY

INDEX

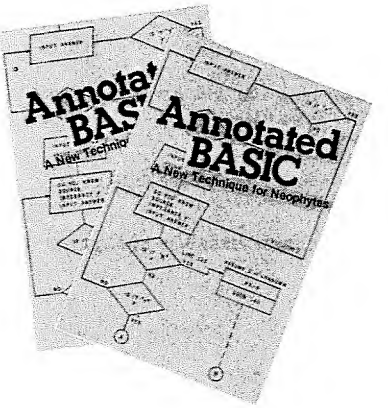
Volumes 1–10

INDEX COMPILED BY NAN MCCARTHY

Wayne Green Books

Annotated BASIC

A New Technique for Neophytes



Put your BASIC knowledge to work for you with this 2-volume set of TRS-80 Level II BASIC programs.
Gain a better understanding of the elements and techniques involved in programming.
Annotated BASIC's uniquely designed format breaks each program down for you to include:

- initial documentation and instruction
- definitions of New BASIC Concepts
- flowchart
- annotations of sections, showing how each part fits into the whole, and explaining why certain BASIC commands are chosen over similar ones.

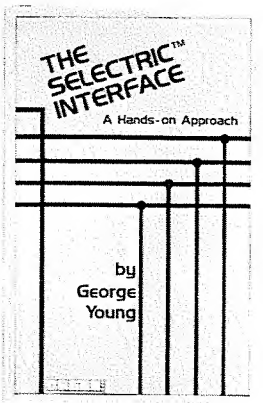
Using the programs as they are or modifying them to sharpen your programming skills, **Annotated Basic** is a helpful tool for any BASIC programmer.

Volume 1	ISBN 0-88006-028-X	152 pages	**$10.95**
Volume 2	ISBN 0-88006-037-9	136 pages	**$10.95**

WAYNE GREEN BOOKS
Division of Wayne Green Inc.
Peterborough, NH 03458
FOR TOLL FREE ORDERING:
1-800-258-5473

Wayne Green Books

Daisy wheel quality without daisy wheel expense.



You need the quality print that a daisy wheel printer provides but the thought of buying one makes your wallet wilt. *Selectric™ Interface*, a step-by-step guide to interfacing an IBM Selectric I/O Writer to your microcomputer, will give you that quality at a fraction of the price. George Young, co-author of *Microcomputing* magazine's popular "Kilobaud Klassroom" series, offers a low-cost alternative to buying a daisy wheel printer.

Selectric Interface includes:

- step-by-step instructions
- tips on purchasing a used Selectric
- information on various Selectric models, including the 2740, 2980, and Dura 1041
- driver software for Z80, 8080, and 6502 chips
- tips on interfacing techniques

With *Selectric Interface* and some background in electronics, you can have a high-quality, low-cost, letter-quality printer. Petals not included.

ISBN 0-88006-051-4 128 pages **$12.97**

WAYNE GREEN BOOKS
Division of Wayne Green Inc.
Peterborough, NH 03458

FOR TOLL-FREE ORDERING:
1-800-258-5473

*TRS-80 is a trademark of Radio Shack division of Tandy Corp.

The real value of your computer lies in your ability to use it. The capabilities of the TRS-80* are incredible if you have the information which will help you get the most from it. Little of this information is available in your instruction books.

The *Encyclopedia for the TRS-80* will teach you how to get the most from your computer. In addition to a wealth of programs which are ready for you to use, reviews of accessories and commercially available programs, you will also learn how to write your own programs or even modify commercial programs for your own specific use.

The *Encyclopedia* is packed with practical information, written and edited for the average TRS-80 owner, not the computer scientist. You will find it interesting and valuable.

Wayne Green
Publisher

$10.95 ISBN 0-88006-049-2